Qore Programming Language Reference Manual

0.8.11.1

Generated by Doxygen 1.8.8

Wed Oct 22 2014 11:05:51

Contents

Chapter 1

Qore Language Reference Manual

Chapter 2

Introduction

2.1 Introduction to Qore

The Qore programming language is a powerful, thread-capable, embeddable weakly-typed language with optional strong typing and procedural and object-oriented features designed for anything from quick scripting to complex multithreaded, network-aware application development to embedded application scripting. Qore was initially designed to facilitate the rapid implementation of sophisticated interfaces in embedded code in an enterprise environment, and has since grown into a general-purpose language as well.

Qore exports a C++ API to allow programs or libraries to embed Qore code; this manual documents Qore's user-level features, for more information about Qore's C++ API, see the `Qore home page`.

Flexible character encoding support is also built-in to Qore strings, and automatic character encoding conversions are supported, enabling correct behavior when working in an environment with mixed character encoding requirements (see Strings and Character Encoding).

Qore includes the following design points:

Support for Embedded Logic
Qore was designed to support embedding logic in applications; this also applies to applications written in Qore as well as applications using the Qore library's public C++ API. By using the Program class, discrete objects can be created and destroyed at will containing embedded code to extend or modify the behavior of your application in user-defined ways. The Program class allows the capabilities of embedded code to be arbitrarily restricted as well.

Thread Safety and SMP Scalability
All elements of Qore are thread-safe, and the language in general has been designed with SMP scalability in mind. The internal design and implementation of Qore favors multithreaded performance over single-threaded performance, so multithreaded Qore programs can count on an efficient and stable execution platform, and do not have to limit themselves to a subset of Qore's functionality (see Threading). Additionally, Qore includes optimizations designed to reduce the number of SMP cache invalidations that provide a substantial performance boost on SMP machines.

Qore supports deadlock detection in complex locking scenarios and will throw an exception rather than allow an operation to be performed that would cause a deadlock. Furthermore, Qore's threading primitives detect threading errors and throw exceptions in these cases as well.

Database Integration and DBI Layer
Retrieving, comparing, and manipulating data in a consistent manner from heterogenous database types is made possible by Qore's built-in database integration. Qore was designed with a database independent interfacing (DBI) layer, providing a standard interface for Qore programs to access any database supported by a Qore DBI driver (see the Datasource class). Qore now supports a very high level API in `SqlUtil`, which provides facilities for automatic schema management and programmatic SQL generation as well as data synchronization between heterogenous database types and more.

Function and Class Library
Qore's basic functionality covers areas such as: POSIX-compliant command-line parsing (ex: GetOpt class), strong

encryption and digest calculation, thread synchronization (ex: Queue class, Mutex class, Condition class, etc), working with files (File class), socket, HTTP, and higher-level protocol communication (Socket, HTTPClient, Ftp↩ Client classes, optionally with TLS/SSL encryption), support for dynamic embedded application logic (Program class). Additionally, Qore's functionality is extended with modules delivered separately from the Qore library (see Module Description or `Qore's home page` for more information.

Familiar Syntax

Qore syntax is similar to other programming languages, allowing new programmers to rapidly come up to speed in Qore. Qore borrows features from languages such as: C++ (ex: multiple inheritance, exception handling, static methods), Java (ex: the synchronized keyword, the instanceof operator, object and class implementation), `Perl` (ex: the foreach statement, splice, push, pop, chomp, trim operators, perl5-compatible regular expressions, and more), the D Programming Language (the on_exit, on_success, and on_error statements provide exception-aware functionality similar to D's `scope(exit)`, `scope(failure)`, allowing exception-aware cleanup code to be placed next to the code requiring cleanup), Python (range(), xrange(), and the RangeIterator class), and others, also with many features unique to Qore. Furthermore, Qore supports closures (including binding local variables in the closure in a way that is safe to use even in multithreaded contexts) and features for advanced list processing (map, foldl, foldr, and select).

Qore's operators are designed to produce the expected results for the programmer even when data types are mixed, a feature meant to further flatten the learning curve for new programmers.

Additionally, every effort is made to find and fix bugs in Qore before every release; for example `valgrind` is used on Linux and OSX and dbx on Solaris to check for memory leaks and memory errors. Particular attention is paid to thread-safety of all Qore components as well; Qore aims to be a stable platform for enterprise development supporting both quick script-based solutions and as an encapsulating language for embedding (and protecting) code in a server application.

Chapter 3

Language Overview

A Qore program is composed of a series of declarations, statements, function definitions, and/or class definitions. Non-block statements are terminated by a semi-colon ";". Block statements are grouped by using curly brackets ("{" and "}"), as in C, C++, Java, and Perl.

Programmers familiar with C, C++, Java, and/or Perl should find the standard Qore syntax intuitive and should be productive fairly quickly with the language. However Qore has unique features that differentiate it from other languages, and these features must be mastered in order to leverage the full power of Qore.

Qore programs/scripts are free form. Formatting does not affect the execution of the program; formatting is at the discretion of the programmer and should be used to enhance readability and clarity of the source code.

Qore was created as a weakly typed language. That means that variables (without type restrictions) can hold values of any type and functions (without a return type restriction or parameter type descriptions) can return any data type and take arguments of any type. Furthermore list elements can be of any type (they do not have to be uniform), and multidimensional lists can have a different number of elements in each list. The same type flexibility holds true of hashes, objects, and all combinations of container types.

Qore also allows variable, parameter, class member, and return types to be declared, so that APIs can be formally defined or the programmer can decide to declare types to catch more errors at parse time (which is often preferable to discovering a type error at runtime).

Qore can be used as a traditional function-based scripting language or as a pure object-oriented language, where the application is defined as a class. Aside from traditional local and global variables, constants, and functions, Qore also supports nested namespaces, classes, multiple inheritance, overriding base class constructor arguments, public and private members and methods, static class methods, and static class variables.

All elements of Qore are designed to work together: database access, socket communication, embedding logic in subprograms, regular expressions, operators, functions, and all other elements are thread-safe and built on an execution engine that was designed for SMP scalability.

Qore automatically converts data types when necessary when evaluating operators. The goal is to provide the expected result for the programmer without requiring the programmer to explicitly convert data types. Please see Operators for more information.

Qore supports signal handling by executing Qore-language signal handlers in special single-handling thread.

UNIX operating systems allow an executable script to specify their interpreter. This is done by setting the first line in the program to a special string indicating the location of the Qore binary. For the purposes of this document, the location for the Qore binary is assumed to be `/usr/bin/qore`. The first line of Qore scripts in this case should look as follows:

```
#!/usr/bin/qore
```

If another installation directory is used (such as `/usr/local/bin`), then the correct path must be reflected in the first line of the Qore script.

Qore convention dictates that Qore script file names end with `".q"`.

Chapter 4

Environment Variables

This section will outline the environment variables that are used by Qore.

Qore Environment Variables

Environment Variable	Description
QORE_AUTO_MODULE_DIR	This environment variable should contain a colon-separated list of directories which will be searched for Qore modules when Qore starts. If any modules are found in any of these directories, they are loaded automatically before any parsing starts.
QORE_MODULE_DIR	This environment variable should contain a colon-separated list of directories which will be searched when modules are loaded with the %requires parse directive
QORE_INCLUDE_DIR	This variable should be a colon-separated list of directories where the Qore binary should look for include files
QORE_CHARSET	If this variable is set, then the default character encoding name for the process will be the value of this variable. This variable takes precedence over the LANG variable, but can be overridden by the command line using option -charset (see Strings and Character Encoding for more information on this option)
LANG	If this variable is set and includes a character encoding specification, then, if the QORE_CHARSET variable is not set (and no character encoding was specified on the command line), this character encoding will be the default for the process.
TZ	On UNIX systems, this environment variables points to the zoneinfo file that contains the definition of the current time zone; for more information, see UNIX Time Zone Handling

Chapter 5

Conditional Parsing and Parse Defines

Qore supports conditional parsing with parse defines similar to the C/C++ preprocessor. In the current version of Qore, the implementation is very simple; only the following parse directives are supported: %define, %else, %endif, %ifdef, and %ifndef.

Basically, the above allow for the existence (or lack thereof) of a parse define to affect which code is parsed into the program at parse time.

Parse defines are defined on the command-line (or through the C++ API when executed in embedded code), as well as created automatically based on sytem options (see Qore::Option for a list of option constants that are also defined as parse defines); all library options (if the option is True, then it is defined as True, if the option is False, then it is not defined at all).

Note that `"Unix"` is defined on all Unix platforms (also on Cygwin), while `"Windows"` is defined on native Windows ports (but not on Cygwin, as this is treated as Unix when compiling, as all Unix features are available).

Additionally, the following options are defined in every program (however they are not yet useful when parsing as the value of parse options cannot be used yet at parse time; only the existence or lack thereof can affect parsing in this version of Qore when parsing at least).

Qore Parse Defines

Define	Value
`QoreVersionString`	Version string for the Qore library
`QoreVersionMajor`	Major version for the Qore library
`QoreVersionMinor`	Minor version for the Qore library
`QoreVersionSub`	Sub version for the Qore library
`QoreVersionBuild`	Build version for the Qore library
`QoreVersionBits`	32 or 64 depending on the library target
`QorePlatformCPU`	The CPU targeted by the library
`QorePlatformOS`	The OS targeted by the library

Additionally, only if the Qore library was compiled with debugging support, the following parse define is present (otherwise it is not defined):

Qore Optional Parse Defines

Define	Value
`QoreDebug`	True

See also

Qore::Option for a list of option constants that are also defined as parse defines

Here is an example of using parse defines in a program:

```
%ifndef HAVE_TERMIOS
printf("This program requires UNIX TermIOS features to be present; it does not run on platforms
       without this feature (current platform: %s); exiting...\n", Qore::PlatformOS);
```

```
exit(1);
%endif
```

Furthermore, parse defines can be manipulated in embedded code using the following functions:

- Qore::Program::define()
- Qore::Program::getDefine()
- Qore::Program::isDefined()
- Qore::Program::undefine()

Chapter 6

Module Description

6.1 Module Overview

Qore modules allow the Qore language to be extended at run-time.

To load a module at parse time (normally required for most modules), use the %requires or %try-module parse directive. If the named feature is not already present in Qore, Qore looks for a module with this name in the directories listed in the `QORE_MODULE_DIR` environment variable (see Environment Variables).

Use the load_module() function to load Qore modules at run-time; however, note that any module providing parse support (classes, constants, functions, etc) must be loaded at parse time using the %requires or %try-module parse directive.

From Qore 0.7.1 onwards, you can specify a comparison operator (one of < <=, =, >=, or >) and version information after the module name as well. Version numbers are compared via integer comparisons of each element, where elements are separated by a '.'. If one of the versions does not have as many elements as another, the missing elements are assumed to be '0' (i.e. version `"1.0"` compared with version `"1.0.1"` will be extended to `"1.0.0"`).

Also note that DBI drivers are loaded on demand by the Qore::SQL::Datasource and Qore::SQL::DatasourcePool classes.

There are two types of modules: Binary Modules and User Modules.

At the time of writing this documentation, the following modules exist for Qore:

Modules Provided With Qore

Type	Module	Description
user	`CsvUtil`	Provides code to help parse CSV or other structured text files and also to easily generate such files
user	`HttpServer`	Provides a multi-threaded HTTP server and request handler APIs; provides the infrastructure for server-side HTTP services
user	`MailMessage`	Provides supporting classes for the `Pop3Client` and `SmtpClient` modules; mail message serialization and deserialization, attachment handling

user	Mapper	Provides data mapping classes and interators
user	Mime	Provides MIME definitions and functions; MIME type lookups, MIME encoding and decoding functions, MIME multipart handling, etc
user	MysqlSqlUtil	Provides a high-level DB-independent API for working with MySQL database objects; loaded automatically by the SqlUtil module when working with MySQL databases for automated schema management, programmatic DB access, schema and data synchronization, and more
user	OracleSqlUtil	Provides a high-level DB-independent API for working with Oracle database objects; loaded automatically by the SqlUtil module when working with Oracle databases for automated schema management, programmatic DB access, schema and data synchronization, and more
user	Pop3Client	Provides POP3 client functionality; provides an API to retrieve email messages from a POP3 server
user	PgsqlSqlUtil	Provides a high-level DB-independent API for working with PostgreSQL database objects; loaded automatically by the SqlUtil module when working with PostgreSQL databases for automated schema management, programmatic DB access, schema and data synchronization, and more
user	Qorize	Provides basic support for automatically-generating Qore code from data
user	RestClient	Provides a simple API for communicating with HTTP servers implementing REST services
user	RestHandler	Provides an easy to use interface to the Qore HttpServer module for implementing server-side REST services

user	Schema	Provides automatic schema management functionality as a meta-layer for SqlUtil's medium and low-level schema management functionality
user	SmtpClient	Provides SMTP client functionality; provides an API for sending emails via an SMTP server
user	SqlUtil	Provides a high-level DB-independent API for working with databases; for automated schema management, programmatic DB access, schema and data synchronization, and more
user	TableMapper	Provides a data mapping functionality and iterator support using SqlUtil and Mapper to map arbitrary data to an SQL table target
user	TelnetClient	Provides Telnet client functionality
user	Util	Provides a some miscellaneous generally useful routines; often used by other user modules for example
user	WebSocketClient	Provides an event-driven client API for connecting to WebSocket servers
user	WebSocketHandler	Provides an interface to the Qore HttpServer module for implementing server-side WebSocket services
user	WebSocketUtil	Provides common client and server code for implementing WebSocket protocol services in Qore
user	WebUtil	Provides higher-level classes for implementing more complex web services in Qore; works with the HttpServer module

Modules Provided Separately

Type	Module	Description
binary	freetds	Provides a FreeTDS-based DBI driver for communicating with Sybase and MS SQL Server databases
binary	fsevent	Provides an event-driven filesystem event API
binary	glut	Provides GLUT functionality
binary	json	Provides JSON and JSON-RPC client functionality, also provides the following user modules: - JsonRpcHandler: provides infrastructure for implementing JSON-RPC server-side services using the HttpServer module
binary	linenoise	Provides a readline-like API to Qore under a permissive license
binary	mysql	Provides a MySQL / MariaDB / Percona DBI driver
binary	oracle	Provides an Oracle DBI driver, providing many advanced features such as support for named types and collections, advanced queuing, etc, also provides the following user modules: - OracleExtensions: provides infrastructure for SQL tracing
binary	opengl	Provides an OpenGL API to Qore
binary	openldap	Provides an OpenLDAP API to Qore
binary	pgsql	Provides a PostgreSQL DBI driver
binary	sqlite3	Provides an SQLite3 DBI driver
binary	ssh2	Provides SSH2 and SFTP functionality, also provides the following user module: - SftpPoller: provides event-driven support for polling an SFTP server for new data
binary	sybase	Provides a Sybase DBI driver
binary	sysconf	Provides sysconf, pathconf, and confstr APIs
binary	tibae	Provides TIBCO ActiveEnterprise(TM) (TIBCO, Inc) functionality
binary	tibrv	Provides TIBCO Rendezvous(R) (TIBCO, Inc) functionality
binary	uuid	Provices an API for generating UUIDs
binary	xml	Provides XML (SAX and DOM parsers), XPath, XML-RPC, SOAP client and server, etc functionality, also provides the following user modules: - SoapClient: provides an easy to use API for making requests to SOAP servers - SoapHandler: provides infrastructure for implementing SOAP server-side services using the HttpServer module - WSDL: provides underlying web service/WSDL support for the SoapClient and SoapHandler modules

binary	xmlsec	Provides xmldsig and xmlenc functionality
binary	yaml	Provides YAML functionality, also provides the following user modules: - DataStreamClient: provides a DataStream client API extending the RestClient - DataStreamRequestHandler: provides a DataStream server-side handler API extending the RestHandler - DataStreamUtil: provides underlying DataStream client and server protocol support - YamlRpcClient: provides an API for easily making YAML-RPC calls over the network - YamlRpcHandler: provides infrastructure for implementing YAML-RPC server-side services using the HttpServer module

Other Modules Provided Separately

Type	Module	Description
binary	asn1	Provides some ASN.1 functionality; stable but only partial ASN.1 functionality is provided, uses very old APIs
binary	db2	Provides an IBM DB2 driver; stuck in protype phase due to lack of development
binary	ncurses	Provides curses APIs; stable and works, but uses old APIs, needs updating
binary	qt4	Provides Nokia (formerly Trolltech) QT4 APIs for GUI development; this module works for a subset of the QT API, but due to the need for complete object lifecycle control, it's not recommended for use until we can implement a new garbage collector for Qore objects created from external modules
binary	tuxedo	Provides Oracle (ex Bea) Tuxedo functionality; this module has not been updated for some time, uses old APIs

6.2 Binary Modules

Binary modules are written in C++ and delivered in binary form. They must conform to the Qore Module API and have the file extension "*.qmod". Binary modules normally depend on other shared libraries and therefore can only be loaded if the libraries they require are present on the system and can be found in the library path.

Binary modules are merged into Program objects in the same way as the static system namespace objects are imported. It is possible to import a binary module and be able to use only part of its functionality, for example, if the importing Program cannot access the filesystem, and the module has functions that access the filesystem, only

the functions that access the filesystem will not be available. In user modules, the functional domain is set on the module level and not on the individual function or method level as with builtin objects, so user modules are either completely imported or not at all.

Please note that as of version 0.7.0 onwards, the source code for binary modules has been split from the main Qore library source code into separate projects; see Qore's home page for more information.

6.3 User Modules

User modules are written in Qore and delivered in source form. They must have the extension `"*.qm"`.

User modules will have their own dedicated Program object; the Program object is created automatically when the module is loaded and initialized. The module's Program object is created with locked parse options as follows:

- PO_NO_TOP_LEVEL_STATEMENTS: modules provide API services and are not designed to be executed at the top-level. Put any initialization code in the module's `init` closure
- PO_REQUIRE_PROTOTYPES: to ensure that module's APIs are transparent, all method and function declarations must include parameter and return types
- PO_REQUIRE_OUR: this parse option is meant to ensure that typos in variable names are caught at parse time and to ensure transparency of variable types in the module's source code
- PO_IN_MODULE: this parse option is set so that module code is recognized by the system when parsing modules

When parsing module code, the default warning mask is set to WARN_MODULES, and any warnings are treated as errors. Furthermore, any restrictions that the importing Program object has will be added to the module's Program object.

Any namespaces, classes, constants, functions, and global variables declared as public will be exported into the importing Program object; all other declarations will be private to the module's Program object. Note that the root namespace of a module Program is always public by default.

Note that global variables exported from a module's Program object are exported as references; each global variable declared in a module is unique, and they reside in the module's Program object.

Furthermore, when using an environment with multiple Program objects, if a user module has already been loaded and initialized, then it's functional domain mask is compared against any importing Program object's restrictions; if the module uses functionality that is not allowed in the importing Program object, then an exception is raised and the module is not imported.

Also note that the Qore::Program::constructor() applies a mask to the parse option mask option when Program objects are created in a user module; in this case the parse options passed to the child Program object are masked with the current parse options in the user module, and they are locked so that they cannot be made less restrictive. This is to prevent user modules from circumventing functional restrictions imposed by parse options.

6.3.1 User Module Declarations

User modules are declared with a special syntax in Qore:

User Module Declaration Syntax

```
module name {
version = "version string";
desc = "description string";
author = "author string";
[url = "URL string";]
[license = "license string";]
[init = initialization closure;]
[del = deletion closure;]
}
```

Module properties are as follows:

- `version:` (required) must be assigned to a string giving the version of the module
- `desc:` (required) must be assigned to a string giving a text description of the module
- `author:` (required) must be assigned to a string giving the module's author's name
- `url:` (optional) if present, must be a string giving the URL of the module
- `license:` (optional) if present, must be a string giving the license of the module
- `init:` (optional) if present, must be a closure that will be executed when the module is loaded; this can be used to initialize the module, for example
- `del:` (optional) if present, must be a closure that will be executed when the module is unloaded; this can be used to stop running services or free resources managed and still allocated by the module, for example

Note

- any unhandled exceptions in the `init` closure will cause the module to fail to load
- unhandled exceptions in the `init` and `del` closures are displayed on stdout

6.3.2 The "public" Keyword

Only objects defined with the **`public`** keyword are made available in Program objects importing the user module. All other declarations and definitions are private to the module.

The **`public`** keyword also affects inheritance in child Program objects as well as controlling exported symbols from user modules.

The **`public`** keyword must be used with the following declarations in the module to export them:

- namespaces: namespaces themselves must be declared **`public`** in order for any of their contents also to be exported (it is an error to try to declare public members of a module-private namespace). ex:

```
public namespace MyNamespace { ... }
```

- classes: classes not declared **`public`** will not be exported; there is no way to export part of a class; either the entire class is exported or it is not. ex:

```
public MyNamespace {
    public MyClass { ... }
}
```

- constants: constants must be declared **`public`** to be exported; ex:

```
public namespace MyNamespace {
    public const MyConst = 100;
}
```

- functions: function variants must be declared **`public`** in order to be exported. ex:

```
public MyNamespace {
    public int sub my_func() { ... }
}
```

- global variables: only global variables declared **`public`** will be exported. ex:

```
public namespace MyNamespace {
    public our int $OurInt;
}
```

Note

Global variable declarations in a namespace declaration cannot be initialized at the point they are declared, also, since Qore::PO_NO_TOP_LEVEL_STATEMENTS is set for user module Program objects, global variables also cannot be initialized at the top-level. Use the `init` closure to initialize the module and any global variables requiring initialization.

User modules are only imported into a Program if the importing Program can use all of the capabilities used in the user module. It is not possible to partially import a user module (in contrast with binary modules, which can be imported even if they expose functionality not allowed in the importing Program, however that functionality will not be available in that case). User module Program objects have a functional domain attribute set on the Program level, so either a user module is imported in its entirety or not at all.

6.3.3 Module Example

Here is an example declaring user module `"foo"` version `"1.0"`:

```
module foo {
    version = "1.0";
    desc = "test module";
    author = "Foobar Industries, inc";
    url = "http://example.com";
    license = "MIT";

    init = sub () {
        Foo::initialized = True;
        Bar::OurBool = False;
    };

    del = sub () {
        print("goodbye, cruel world!\n");
    }
}

# do not use "$", assume local scope for variables
%new-style

# nothing in namespace ::Foo is exported
namespace Foo {
    # inline global variable declarations cannot be initialized when declared
    our bool initialized;

    class SuperClass {
    }
    class NotSoGreatClass {
    }
    class UnstableClass {
    }
}

# public members of namespace ::Bar are exported
public namespace Bar {
    # Bar::SomeClass is exported
    public class SomeClass {
    }

    # Bar::add(int, int) is exported
    public int sub add(int x, int y) { return x + y; }

    # Bar::OurBool is exported
    public our bool OurBool;

    # Bar::PrivateClass is not exported
    class PrivateClass {
    }
}
```

Since

Qore 0.8.4 user modules are supported

6.3.4 Program Scope in Object-Oriented Programs Using User Modules Providing Their Own Threads

When using the %exec-class parse directive, the application object will go out of scope as soon as the constructor terminates (unless there are valid scope-extending references to the application object, such as making an assignment of `$self` to a global variable in the contructor).

Therefore when using a module that provides services in its own threads (such as, for example, the `HttpServer` module), it's important to make sure that the application object does not go out of scope while non-static method call references to the application object are needed by active threads inthe user module (for example, non-static method call references passed as callbacks to the `HttpServer` module, etc).

This also applies to call references to non-static methods passed to set_signal_handler().

If a module thread tries to use a callback or closure in a Program that has already gone out of scope, an `OBJECT-ALREADY-DELETED` exception (`"attempt to access member of an already-deleted object"`) is thrown.

Therefore in such cases it's best to wait for all threads in any modules to terminate before allowing the application object's constructor to terminate.

For an example of this, see the example program `examples/httpserver.q` where the main Program calls HttpServer::waitStop() before exiting the application object's constructor for this reason.

Chapter 7

Include Files

A Qore program can include other program code to be used and executed in the current program by using the %include directive. The %include directive must be the first text on the line, and the file name to include must follow. All text on the line after the %include directive will be used for the file name to include. The file name should not be quoted.

Here is an example:

```
#!/usr/bin/qore
%include /usr/qore/lib/functions.ql
```

After this, any variable, function, namespace, constant, or object declared in the above file can be used by the Qore program.

The `QORE_INCLUDE_DIR` environment variable determines the search path for include files (see Environment Variables).

Chapter 8

Identifiers

Qore identifiers must start with an alphabetic character, and then may contain any number of alphabetic, numeric, or "_" characters. There is no length limit on Qore identifiers.

All Qore identifiers are case-sensitive, therefore the identifier `hello_there` is not the same as `Hello_There` or `HELLO_THERE`.

The following are examples of valid Qore identifiers:

Examples of Valid Qore Identifiers

Identifier	Description
`i`	Simple one-character identifier
`foo21`	Identifier with number
`this_is_a_long_identifier`	Long identifier with underline characters
`Total_318`	Identifier with underline and number
`AVeryBigNumber`	Mixed case identifier name
`CAPS`	Identifier in all capital letters

The following are invalid identifiers:

Examples of Invalid Qore Identifiers

Identifier	Description
`1a`	Does not start with an alphabetic character
`this-and-that`	Contains "-" characters
`Start#10`	Contains "#" character

Chapter 9

Comments

Comments are allowed in Qore scripts; line comments are preceded by a hash "#", and block comments are made C-style, ex:

```
# this is a line comment
/*
   this is a block comment
*/
```

For line comments, all text following a hash until the end-of-line character "\n" is considered a part of the comment.

For block comments, all text in the block comment is ignored by the parser.

Here is an example Qore script containing comments:

```
#!/usr/bin/env qore
#
# these are line comments
# another line comment

/*
   --- this text is in block comments
   print("hello"); <- this won't get executed
   --- because it's in the block comment
*/
```

Chapter 10

Variables

Unless parse option %allow-bare-refs or %new-style are set, variables are Qore identifiers prefixed by a "$" sign, similar to Perl. If a variable is declared without any type restriction, then it is assumed to have type any. In this case, variables so declared can hold any data type.

10.1 Special Variables

A few variables are set by the Qore language during the execution of Qore programs. These are normal variables that can be reassigned to other values by the user if necessary.

Special Qore Variables

Variable	Scope	Data Type	Explanation
$argv	Local	List	automatically assigned local variable containing the list of function or method arguments that were not assigned to parameter variables (see Implicit Argument References for more information)
Qore::$ARGV	Global	List	script command-line arguments (use the Qore::GetOpt class to parse command-line arguments)
Qore::$QORE_ARGV	Global	List	complete qore command-line arguments
Qore::$ENV	Global	Hash	UNIX program environment

Note

- As of version 0.5.0, $STDERR and $STDOUT have been removed from Qore. Use the I/O constants Qore::stderr, Qore::stdout, and Qore::stdin constants of the Qore::File class instead.
- As of version 0.8.4, global variables are namespace members; if a namespace path is not declared for the global variable, then the global variable will reside in the root namespace

10.2 Variable Declarations and Lexical Scope

Unless the %assume-local parse directive is used, variables not in a parameter list automatically have global scope unless the first reference is prefixed with The "my" Keyword. Variable names in a parameter list are always local to their associated function, method, or catch block. Global variables can be explicitly declared with The "our" Keyword. The The "our" Keyword keyword is required if the parse option %require-our (`-O` or `--require-our` command-line option) is set for the parent program. See Parse Option Constants for more information.

When the %assume-local parse directive is used, variables without an explicit scope declaration (i.e. The "my" Keyword or The "our" Keyword) are assumed to be local variables.

Variables may be assigned any value unless restricted with a type declaration. If no type declaration is given, then the variable is assumed to be type any. Note that type declarations are required for all variables (and for function and method parameters and class members) when the %require-types parse option is set.

Local variables are not shared between threads (local variables have a distinct value in each thread, with some very particular exceptions), however global variables are. See Threading (and in particular Threading and Variables) for more information.

Global variables are members of namespaces; if a global variable is declared anywhere outside a namespace declaration, and a namespace path is not given, then the global variable will be assumed to be in the root namespace.

For example (in the following script, the The "our" Keyword keyword is optional unless %require-our is set):

```
#!/usr/bin/qore
#
# variable scoping example

our int $a = 1;                    # this is a global variable
our (string $b, any $c, hash $d);  # list of global variables

if ($a == 1) {
    my int $a = 2;
    my (string $b, any $c);
    # $a, $b, and $c are local variables,
    # the use of which will not affect the
    # global variables of the same name
    print("local a = %d\n", $a);
}

print("global a = %d\n", $a);
```

The first print() statement will output:

```
local a = 2
```

The second print() statement will output:

```
global a = 1
```

Note

If parse option %allow-bare-refs is set, then variable references **must** be made without the "$" character.

10.3 Local Variables

Local variables are local to the block in which they are declared; they are also thread-local, meaning that each thread will have its own value of a local variable.

Local variables are generally accessed without any mutual-exclusion locking (with the exception of local variables bound in closures and local variables that have a reference taken of them).

Note

Declaring a variable with **my** at the top-level of a program creates a local variable with global scope; in effect this is a global thread-local variable; see Threading and Variables for more information.

10.3.1 The "my" Keyword

Variables declared with **my** are always local variables.

my Example

```
my int $i = 1;
```

The **my** keyword is not required if the %assume-local parse directive is set (note that this parse directive is also set by %new-style). In this case, all variables are assumed to be local unless explicitly declared with our.

10.4 Global Variables

10.4.1 The "our" Keyword

Variables declared with **our** have global scope and are subject to mutual exclusion locks with each access. Therefore even in single-threaded programs, it's more efficient to use local variables (even local variables with global scope - where a local variable is declared in the top-level block) if the value of the variable does not need to be shared among other threads.

our Example

```
our int $i = 1;
```

Note that the **our** keyword is required when declaring a global variable if the parse option %require-our (`-O` or `--require-our` command-line option) is set for the parent program. See Parse Option Constants for more information.

Unlike local variables, global variables are members of namespaces; if a namespace path is not given, then the global variable will be assumed to be in the root namespace. Global variables declared in namespaces cannot be initialized at the same time as the declaration, but instead have to be initialized elsewhere.

When defining a user module, the **our** keyword can be preceded by public, which means that the global variable will be available (imported) in the Program object importing the module. See public for more information.

Chapter 11

Basic Data Types

The following are the basic data types in Qore (see Container Data Types and Code Data Types):

Basic Data Types

Type	Description	Example	Default Value
Boolean	True or False value	True	False
String	A sequence of characters with a character encoding	`"string"`	Empty string (i.e. `""`)
Integer	A 64-bit signed integer	`1`	`0`
Float	A double-precision floating-point number	`1.00023`	`0.0`
Number	An arbitrary-precision number	`5.↩ 23928173726123e50n`	`0.0n`
Date	absolute (with an associated time zone) or relative date/time values, both with resolution to the microsecond	`2010-05-10T18:35↩ :21.001456-07:00`	`1970-01-01Z`
Binary	An opaque binary object	`<23deadbeef>`	an empty object of size 0
NULL	Corresponds to a `NULL` value in a database query (not equivalent to NOTHING	Qore::NULL	Qore::NULL
NOTHING	Represents the state of a variable having no value or function returning no value (not equivalent to NULL)	Qore::NOTHING	Qore::NOTHING

11.1 Boolean

Description:

The Boolean type can have two values, True and False.

When converting other types to a Boolean, Qore uses Perl-style boolean conversion, based on an interpretation of the source value and taking into account its type (for example, when this option is set, an empty string, hash, or list is always Qore::False, whereas if these are not empty they are interpreted as True); this way is considered to be more intuitive by most prorammers, so much so that the original strict mathematical interpretation of boolean values is considered to be a design bug in qore; see %perl-bool-eval for more information.

Immediate Value Example:

Qore::True

Pseudo Class for Type Boolean:

Qore::zzz8boolzzz9

Type Code:

Qore::NT_BOOLEAN

Type Name:

`"bool"`

See also

bool, softbool

11.2 String

Description:

String values are specified with text between double or single quotes. Text between double quotes is subject to interpretation of escape characters; text between single quotes is not with the exception of the single quote character, which is the only character that may be escaped (ex: `'hello \'there\"`).

Strings are assumed by default to have the encoding given by the `QORE_CHARSET` or the `LANG` environment variable (see Environment Variables). If neither of these variables is set, then all strings will be assumed to have `UTF-8` encoding.

For detailed information on Qore character encoding handling, please see Strings and Character Encoding.

It is legal to specify a string literal with newline characters like the following:

```
$str = "this string is
on more than 1 line";
```

See string escape characters for a description of escape characters in double-quoted strings.

Internally, strings are stored as a pointer to a sequence of bytes terminated by a null (or zero byte), an unsigned integer giving the length of the string, and a pointer to an object giving the string's character encoding.

Qore::zzz8stringzzz9::strlen() is a constant-time operation (ie *O(1)*), however if the string has a multi-byte encoding, then Qore::zzz8stringzzz9::length() (returning the length of the string in characters, not bytes) computational complexity is *O(n)* (however if the character encoding is a single-byte encoding Qore::zzz8stringzzz9::length() is also *O(1)*).

Immediate Value Example:

`"this is a string"`

Pseudo Class for Type String:

Qore::zzz8stringzzz9

Type Code:

Qore::NT_STRING

Type Name:

```
"string"
```

See also

- string, softstring
- String Functions

11.3 Integer

Description:

Qore integers are 64-bit signed integers.

Immediate Value Example:

```
100
```

Pseudo Class for Type Integer:

Qore::zzz8intzzz9

Type Code:

Qore::NT_INT

Type Name:

```
"integer"
```

See also

- int, softint
- Qore::MAXINT, Qore::MININT

11.4 Float

Description:

Qore floats are double precision floating-point numbers (C/C++ type *double*), normally a 64-bit value.

Immediate Value Examples:

- `-500.494`
- `2.35e10`

Pseudo Class for Type Float:

Qore::zzz8floatzzz9

Type Code:

Qore::NT_FLOAT

Type Name:

```
"float"
```

See also

- float, softfloat
- Math Functions

11.5 Number

Description:

Qore "number" values are arbitrary-precision numbers as provided by the GMP and MPFR libraries.

Operations with number values are generally slower than those with floats but support far greater accuracy. To give an immediate number value; write an integer or floating-point value and append an "n" to it, designating a "number" value.

In numeric operations where at least one argument is a number type, the other operands will generally be automatically converted to a number type and the result of the operation will also be a number type. When an operator acts on two values of type number, the result of the operation has the precision of the operand with the greatest precision.

Immediate Value Example:

- `-500.494n`
- `2.35e10n`

Pseudo Class for Type Number:

Qore::zzz8numberzzz9

Type Code:

Qore::NT_NUMBER

Type Name:

`"number"`

See also

- number, softnumber
- Math Functions

Since

Qore 0.8.6 introduced the number type and integration with the GMP and MPFR libraries

11.6 Date

Description:

Qore date/time values have date and time components supporting a resolution to the microsecond and can be either absolute or relative.

Immediate Value Examples:

- `2012-02-17T19:05:54+01:00`

Pseudo Class for Type Date:

Qore::zzz8datezzz9

Type Code:

Qore::NT_DATE

Type Name:

`"date"`

See also

- Date and Time Functions for a list of functions related to date/time processing
- Date/Time Arithmetic
- Time Zone Handling

11.6.1 Absolute Date/Time Values

Absolute date/time values specify a specific point in time in a certain time zone, such as January 1, 2005 10:35:00 +01:00. They are stored interally as a 64-bit signed offset from the Qore epoch (`1970-01-01Z`), a non-negative 4-byte integer for microseconds, and a pointer to a time zone description object that provides the UTC offset and daylight savings time information (see Time Zone Handling for more information). Note that all absolute date/time values in Qore are stored internally in UTC and are converted for display purposes to the representation of wall time in their tagged time zone.

Absolute date/time values can be specified with a syntax based on `ISO-8601` date formats as follows:

```
YYYY-MM-DD[THH:mm:SS[.n*]][Z|[+-]HH[:mm[:SS]]]
```

Note that if no time zone information is given, the local time zone will be assumed. If a time zone UTC offset is given, it is given in units of time east of UTC (i.e. `+05`:00 means five hours east of UTC).

Or an alternative format (with a `'-'` instead of a `'T'` to separate the time component):

```
YYYY-MM-DD[-HH:mm:SS[.n*]][Z|[+-]HH[:mm[:SS]]]
```

for example, for just the date without a time component (assumed to be midnight on the given date in the local time zone):

- `2010-05-26`

for just the date in UTC, without a time component:

- `2010-05-26Z`

or, for just the time, without a date component (note that in this case the date component will be set to Jan 1, 1970, in order for time arithmetic to function properly and will also be tagged with the local time zone):

- `20:05:10.458342`

Some further examples (note that the date/time values without a time zone specification here are tagged with the local time zone):

```
prompt% qore -X '2005-03-29-18:12:25'
2005-03-29 18:12:25 Tue +02:00 (CEST)
prompt% qore -X '0512-01-01T01:49:59.002213Z'
0512-01-01 01:49:59.002213 Fri Z (UTC)
prompt% qore -X '2005-03-29'
2005-03-29 00:00:00 Tue +02:00 (CEST)
prompt% qore -X '18:35:26+08:00'
1970-01-01 18:35:26 Thu +08:00 (+08)
```

The year must be a four-digit number, and all other values except microseconds must be two-digit numbers. If microseconds are present, at least one and up to 6 digits may be given after the decimal point. Pad the numbers with leading zeros if the numbers are smaller than the required number of digits. The hour component must be in 24-hour time format. Except for the month and day values, all other values start with 0 (hour = 00 - 23, minute and second: 00 - 59). Any deviation from this format will cause a parse exception.

When a date/time value is converted to an integer or vice-versa, a 64-bit offset in seconds from the start of the "epoch" is used for the conversion. Qore's "zero date" (the start of Qore's "epoch") is January 1, 1970 UTC. When calculating second offsets from this date, a 64-bit integer is used.

Note

The default local time zone for qore is set when the qore library is initialized; see Time Zone Handling for more information.

11.6.2 Relative Date/Time Values (Durations)

Relative dates (durations) are normally used for date addition and subtraction. See Date/Time Arithmetic for more information.

Internally, durations are stored as a set of seven discrete signed integer values, one each for years, months, days, hours, minutes, seconds, and microseconds.

There are 3 different formats understood by the Qore parser for describing literal durations in Qore as follows:

- Single Relative Time Format (ex: `750ms`; Qore-specific)
- Short Relative Time Format (ex: `P3DT21H`; based on `ISO-8601`)
- Long Relative Time Format (ex: `P0001-03-00T00:00:04`; based on `ISO-8601`)

Single Relative Time Format

A single relative date/time value (or a duration) may be specified as follows (note that this format is specific to Qore and not based on `ISO-8601`):

- `<integer><date component specifier>`

Examples:

- `250ms`: 250 milliseconds
- `30s`: 30 seconds
- `2m`: 2 minutes

Such values are recommended to give to functions and methods taking a timeout value as the units are then clear in the source code (whereas if an integer is given, it may not be clear that the function or method expects a value in milliseconds, for example); for example:

```
my any $val = $q.pop(20s);
```

is clearer than the alternative with an argument given as a value in implied milliseconds to the Queue::pop() method:

```
my any $val = $q.pop(20000);
```

Date Specifiers For Single Values For Relative Dates (non-`ISO-8601` syntax)

Component	Meaning	Example	Description

Y	Years	2Y	2 Years
M	Months	3M	3 Months
D	Days	10D	10 Days
h	Hours	15h	15 hours
m	Minutes	25m	25 minutes
s	Seconds	19s	19 seconds
ms	Milliseconds	250ms	250 milliseconds
us	Microseconds	21194us	21194 microseconds

Short Relative Time Format

This and the next duration format for composite relative date/time values are both based on ISO-8601.

This first format has the following syntax:

- PnYnMnDTnHnMnSnu

Each element above is optional, but at least one element must be present. Note that "M" means months when before the "T" and minutes when found after the "T". The other elements are years, days, hours, seconds, and, as an extension to ISO-8601, "u" for microseconds. Additionally, the values may be negative.

Here are some examples (using qore's -X command-line option to evaluate and expression and print out the result):

```
prompt% qore -X 'P1Y3MT4S'
<time: 1 year 3 months 4 seconds>
prompt% qore -X 'PT4M551u'
<time: 4 minutes 551 microseconds>
prompt% qore -X 'P3DT21H'
<time: 3 days 21 hours>
```

Note

the "u" charater is a Qore-specific extension to ISO-8601

Long Relative Time Format

The second ISO-8601-based format for specifing complex durations with multiple time units has the following syntax:

- PYYYY-MM-DDTHH:mm:SS

This format is more limited than the first format, in that all values must be non-negative, and furthermore, all values must be present (although they may be zero).

Here are some examples of the second format (equivalent to the first examples):

```
prompt% qore -X 'P0001-03-00T00:00:04'
<time: 1 year 3 months 4 seconds>
prompt% qore -X 'P0000-00-00T00:04:00.000551'
<time: 4 minutes 551 microseconds>
prompt% qore -X 'P0000-00-03T21:00:00'
<time: 3 days 21 hours>
```

See also

date, softdate

11.7 Binary

Description:

The binary data type is used to hold binary arbitrary binary data. Internally it is represented by a pointer to a memory location for the data and a size indicator.

Binary data can be concatenated with the + and += operators and manipulated with the splice and extract operators.

This data can be manipulated by being written and read from Qore::File, Qore::Socket, Qore::SQL::Datasource, Qore::SQL::DatasourcePool, or Qore::SQL::SQLStatement objects, or converted and parsed to/from base64 encoded strings using the makeBase64String() and parseBase64String() functions, or compressed and decompressed using the compress(), gzip(), bzip2(), etc. functions, and processed by most cryptographic funtions, among others.

Binary objects can be read from a Qore::File object using the Qore::File::readBinary() method and can be written using the Qore::File::write() method. Please see the Qore::File class for more information.

Binary objects can be read from a Qore::Socket object using the Qore::Socket::recvBinary() method and can be written using the Qore::Socket::send() method. Please see the Qore::Socket class for more information.

The Qore::SQL::Datasource, Qore::SQL::DatasourcePool, and Qore::SQL::SQLStatement classes can also be used to read and write binary objects as BLOBs.

Note that this is not an exhaustive list; see the function and class library documentation for more examples.

Immediate Value Example:

```
<0feba023ffdca6291>
```

Pseudo Class for Type Binary:

Qore::zzz8binaryzzz9

Type Code:

Qore::NT_BINARY

Type Name:

```
"binary"
```

See also

binary

11.8 NULL

Description:

This data type represents an SQL `NULL` value and can only be accessed directly as an immediate value with the constant Qore::NULL. Note that Qore::NULL is not equivalent to NOTHING.

Immediate Value Example:

Qore::NULL

Pseudo Class for Type Null:

Qore::zzz8valuezzz9

Type Code:

Qore::NT_NULL

Type Name:

```
"NULL"
```

See also

- Qore::SQL::Datasource
- Qore::SQL::DatasourcePool
- Qore::SQL::SQLStatement
- DBI Functions

11.9 NOTHING

Description:

This special data type represents no value and can only be accessed directly as an immediate value with the constant Qore::NOTHING. Note that Qore::NOTHING is not equivalent to NULL.

Note

The exists operator will return False when given Qore::NOTHING as an argument; for example:

```
prompt% qore -X 'exists NOTHING'
False
```

Immediate Value Example:

Qore::NOTHING

Pseudo Class for Type Nothing:

Qore::zzz8valuezzz9

Type Code:

Qore::NT_NOTHING

Type Name:

```
"nothing"
```

11.10 Data Conversions

Boolean, string, integer, date, floating point, and arbitrary-percision numeric data types can be freely converted from one type to the other, although data loss is possible depending on the conversion (particularly when converting to the boolan type as only two possible values are supported).

The special types NULL and NOTHING are not equivalent and are generally not automatically converted to or from any other type.

When date types are converted from strings, any of the following formats can be used: `"YYYYMMDDHHmmS↩S[.us][Z|+-HH[:MM[:SS]]]"`, `"YYYY-MM-DD HH:mm:SS.us"`, `"YYYY-MM-DDTHH:mm:SS"`, `"Y↩YYY-MM-DDTHH:mm:SS[.us][Z|+-HH[:MM[:SS]]]"`, and most reasonable combinations thereof. If the time zone component is missing, then the local time zone will be assumed (see Time Zone Handling).

When dates are converted to and from integer, floating-point, and arbitrary-percision numeric values, the a 64-bit second offset from January 1, 1970 in the local time zone is used for the conversion. For example

```
int(2006-01-01)
```

gives `1136073600` (regardless of the local time zone the date is in). This is for backwards-compatibility with Qore before time zone support was available; to get the second offset of a date from `1970-01-01Z` (i.e. the true epoch offset), call get_epoch_seconds() instead.

When an expression requires a certain data type and the source data type cannot be converted to the desired data type, the default value for the desired data type will be used. The default values are given in Basic Data Types.

Chapter 12

Container Data Types

Qore supports three types of container types (see also Basic Data Types and Code Data Types):

- lists
- hashes (associative arrays)
- objects (see Object and Classes for more information)

These container types can be combined to make arbitrarily complex data structures. The data type of any element can be any basic type or another aggregate type. The types do not have to be uniform in one container structure.

12.1 List

Description:

Lists (or arrays) are simply ordered containers of values. A list element can be any Qore type (even another list, hash, or object).

Lists are specified by giving expressions separated by commas as follows:

```
$list = (expression, expression [, expression ...]);
```

Here is a concrete example:

```
my list $list = (
    1,
    2,
    "three",
    4.0,
    5e20n,
    6,
    2001-01-15Z,
);
```

Trailing commas can be left on the end of a list (or a hash), making it easier to insert and remove elements at the end of a multi-line list.

List elements are dereferenced using square brackets: `"["` and `"]"`. The first element in a list has index zero.

Example:

```
$element3 = $list[2];
```

The following operators perform special processing on lists:

- elements
- shift
- unshift
- push
- pop
- splice
- []
- +
- +=
- map
- foldl
- foldr
- select

Immediate Value Example:

```
(1, "two", 3.0)
```

Pseudo Class for Type List:

Qore::zzz8listzzz9

Type Code:

Qore::NT_LIST

Type Name:

```
"list"
```

See also

- list, softlist
- List Functions

Note

- Trailing commas can be left on the end of a list (or a hash), making it easier to insert and remove elements at the end of a multi-line list.
- List elements are dereferenced using square brackets: `"["` and `"]"`. The first element in a list has index zero.
- Dereferencing an invalid element (past the end of the list or with a negative number) will return NOTHING
- Use Qore::zzz8listzzz9::iterator() as an easy way to get a list iterator object for the current list

12.2 Hash

Description:

Hashes are containers that associate values to a string key and also preserve key order for consistent data serialization/deserialization. Hash key lookups are case-sensitive and use a hashing algorithm that in the

worst case should provide *O(ln(n))* complexity to execute; in the base case finding a particular key in a hash is executed in constant time (ie *O(1)* complexity).

Hashes are specified using the following syntax:

```
$hash = (
    "key1" : expression,
    "key2" : expression,
    ...,
);
```

Here is a concrete example:

```
my hash $hash = (
    "apple" : 1 + 1,
    "pear"  : "good",
);
```

Trailing commas are ignored (as with lists) to allow for easier insertion and deletion of elements in source code.

Hashes are dereferenced in one of two ways, either using curly brackets: "{" and "}", where any valid Qore expression can be used, or using the dot "." hash member dereferencing operator, where literal strings can be used.

```
$element3 = $hash{"pe" + "ar"};
```

Is equivalent to:

```
$element3 = $hash.pear;
```

and:

```
$element3 = $hash."pear";
```

and:

```
$element3 = $hash.("pe" + "ar");
```

Hash members can have the names of keywords or names that are not valid identifiers, but in this case to dereference them you cannot use the dot operator with a literal string, otherwise a parse error will be raised. Use quotes around the member name when dereferencing hash members with the same name as a qore keyword or other name that is not a valid identifier as follows:

```
$element3 = $hash."keys";
$element3 = $hash{"this-element-1!"};
```

A literal string after the dot "." hash member dereferencing operator must be a valid Qore identifier; therefore if you want to use a hash key that's not a valid identifier, enclose the string in quotes.

If you want to use the result of an expression to dereference the hash, then either the curly bracket syntax must be used or the expression must be enclosed in parentheses.

In the case of using a literal string with the dot operator, keep in mind that the string is always interpreted as a literal string name of the member, even if there is a constant with the same name. To use the value of a constant to dereference a hash, use curly brackets with the constant: ex:

```
$hash{MyConstant}
```

Note that hash keys can also be given by constants (as long as the constant resolves to a string) when using curly brackets.

Immediate Value Example:

```
("key1": 1, "key2": "two", "key3": 3.14159265358979323846264338327950288419 5n)
```

Pseudo Class for Type Hash:

Qore::zzz8hashzzz9

Type Code:

Qore::NT_HASH

Type Name:

```
"hash"
```

See also

hash

Note

- Trailing commas are ignored in immediate hash declarations (as with lists) to allow for easier insertion and deletion of elements in source code.
- Hashes are dereferenced in one of two ways, either using curly brackets: " { " and " } ", where any valid Qore expression can be used, or using the dot "." hash member dereferencing operator, where literal strings can be used. In the case of using a literal string with the dot operator, keep in mind that the string is always interpreted as a literal string name of the member, even if there is a constant with the same name. To use the value of a constant to dereference a hash, use curly brackets with the constant: ex:

  ```
  $hash{MyConstant}
  ```

- Use quotes around the member name when dereferencing hash members with the same name as a qore keyword or other name that is not a valid identifier: ex:

  ```
  $hash."my-tag-1!"
  ```

- Qore hashes preserve key insertion order to support consistent serialization and deserialization to strings (such as XML, JSON, or YAML strings) or data encoding formats, therefore the keys operator and the Qore::zzz8hashzzz9::keys() pseudo-method will always return the hash keys in insertion/creation order
- Dereferencing hash values that are not present in the hash will return NOTHING; to catch typos in hash member names when dereferencing a hash, you can use an object and declare a list of allowed public members in the class definition (in which case dereferencing a non-existant member will cause a parse exception to be thrown, if the object's class is known at parse time, otherwise it will cause a runtime exception), also the Qore::ListHashIterator class allows for hash members to be dereferenced transparently and will throw an exception if a non-existant member name is given (to catch typos).
- There are several pseudo-methods implemented in both the Qore::zzz8hashzzz9 and the Qore::zzz8nothingzzz9 pseudo-classes that provide asy access to special hash iterators:
  - Qore::zzz8hashzzz9::iterator() provides easy access to a hash value iterator; it returns a HashIterator object where the HashIterator::getValue() method returns the current key value
  - Qore::zzz8hashzzz9::keyIterator() provides easy access to a hash key iterator; it returns a HashKeyIterator object where the HashKeyIterator::getValue() method returns the current key string; this is very useful when using a hash as a simulation for a set of strings and quickly iterating the hash with the map operator, for example

- Qore::zzz8hashzzz9::pairIterator() provides easy access to a hash key/value pair iterator; it returns a HashPairIterator object where the HashPairIterator::getValue() method returns a hash with "key" and "value" keys giving the key-value pair for the current iterator position in the hash
- Qore::zzz8hashzzz9::contextIterator() provides easy access to the current element in a hash of lists such as that returned by the Datasource::select() or DatasourcePool::select() methods; it returns a HashListIterator object where the HashListIterator::getValue() method returns a hash of the current row value (which is very useful when iterating query results with the map operator, for example); this class also provides a memberGate() method that allows the iterator object itself to be dereferenced directly as a hash representing the current row (which is useful in foreach statements for example).
- the corresponding pseudo-methods in the Qore::zzz8nothingzzz9 pseudo-class return empty SingleValueIterator objects so that these pseudo-methods can be safely used with *hash (ie "hash or nothing") value types.

12.3 Object

Description:

Qore objects are instantiations of a class. They have members (like a hash; values associated to string keys), and methods. The class definition specifies the methods that run on objects of that class, public and private members, static methods and variables, etc associated with the class (however note that static methods do not run in the scope of an object). See Classes for information on how to create a class in Qore.

Immediate Value Example:

```
new Mutex();
```

Pseudo Class for Type Object:

Qore::zzz8objectzzz9

Type Code:

Qore::NT_OBJECT

Type Name:

"object"

See also

- object
- Object Functions

Note

each Qore type has a "pseudo-class" associated with it (the default is Qore::zzz8valuezzz9); methods from the data type's "pseudo-class" can be run on any value of that type; see "Pseudo Class for Type" headings in Basic Data Types for more information. Pseudo-methods can be overridden in classes; if a class implements a method with the same name and signature as an object pseudo-method, then the class' method will be executed instead of the pseudo-method.

12.3.1 Object Overview

The recommend way to instantiate an object is to declare its type and give constructor arguments after the variable name in parentheses as follows:

```
my class_name_or_path $var_name([argument list])
```

For example (for a constructor taking no arguments or having only default values for the aguments, the list is empty):

```
my Mutex $m();
```

If parse option %new-style is set, the above example declaring a local variable of class Qore::Thread::Mutex would look as follows:

```
Mutex m();
```

Objects can also be instantiated using the new operator as follows.

```
new class_identifier([argument list])
```

For example:

```
my Mutex $m = new Mutex();
```

Or, with parse option new-style:

```
Mutex m = new Mutex();
```

Objects have named data members that are referenced like hash elements, although this behavior can be modified for objects using the memberGate() method. Object members are accessed by appending a dot '.' and the member name to the object reference as follows:

```
object_reference.member_name
```

For more information, see Class Members.

Object methods are called by appending a dot '.' and a method name to the object reference as follows:

```
object_reference.method_name([argument_list])
```

Or, from within the class code itself to call another method from inside the same class hierarchy:

```
$.method_name([argument_list])
```

Or, with parse option new-style, the method call is made without the "$.":

```
method_name([argument_list])
```

For more information, see Object Method Calls.

The object references above are normally variable references holding an object, but could be any expression that returns an object, such as a new expression or even a function call.

Note

Objects are treated differently than other Qore data types; they are only explicitly copied (see Object References for more informaion). Any object instantiated with the new operator will remain unique until deleted or explicitly copied. An explicit copy is made with the copy method, and does not always guarantee an exact copy of the source object (it depends on the definition of the copy method for the class in question).

Objects exist until they go out of scope, are explicitly deleted, or their last thread exits. For detailed information, see Classes.

See also

object

12.3.2 Object References

In Qore objects are treated differently from all other data types in that they are by default passed as arguments to functions and methods by passing a copy of a reference to the object (similar to Java's handling of objects). That means that passing an object to a function that modifies the object will by default modify the original object and not a copy, however reassigning a local parameter variable assigned an object passed as an argument (that is only assigned to a local variable in the calling function) will not result in deleting the object, but rather decrement its scope reference count (note that if the object were created as a part of the call and reassigning the variable would cause the object's scope reference count to reach zero, then the object would be deleted in this case).

Assigning an object to a variable has the same effect; a copy of a reference to the object is assigned to the variable. This results in prolonging the object's scope (by owning a new copy of a reference to the object).

An example:

```
sub test2(any $x) {
   # we can modify the original object like so:
   $x.member = "tree";

   # here we re-assign $x, but since the object is also assigned
   # to $o in the calling function, the object's scope is still
   # valid, and therefore nothing happens so the object
   $x = 1;
}

sub test() {
   my TestObject $o();

   # here we pass a copy of a reference to the object in $o
   test2($o);

   # this will print out "ok\n", because the object is still
   # valid and the member has been set by test2()
   if ($o.member == "tree")
      print("ok\n");
}
# when test() exits, the object in $o will go out of scope
# and be deleted
```

If, however, an object is passed by reference, then the local variable of the called function that accepts the object owns the scope reference of the calling functions's variable.

An example:

```
sub test2(any $x) {
   # we can modify the original object like so:
   $x.member = "tree";

   # here we re-assign $x, and since we own the only scope
   # reference to the object, the object will go out of
   # scope here and be deleted
   $x = 1;
}

sub test() {
   my TestObject $o();

   # here we pass a reference to the object in $o
   test2(\$o);

   # the object has already been deleted in test2() and
   # therefore nothing will be printed out
   if ($o.member == "tree")
      print("ok\n");
}
```

Note

that when parse option %allow-bare-refs is set, then variable references as in the above examples are made without the "$" character.

12.3.3 Object Scope

Objects are automatically deleted when their scope-relevant reference count reaches zero (note that objects can be deleted manually at any time by using the delete operator). Whenever an object is deleted, the object's class' destructor method is run on the object.

The following affect objects' scope:

- variable assignments: an object's automatic scope is prolonged as long as the object is assigned to a local variable
- object method thread launched within the object: if a member function thread was launched from within the object using the background operator, the object's automatic scope is prolonged to the life of the new thread. Object threads started externally to the object (i.e. not directly from an expression with the background operator within a method) will not prolong the scope of the object.

Note

If an object with running threads is explicitly deleted, and this case is not handled in the object's destructor() method (by ensuring that all other running threads terminate gracefully), exceptions will be thrown in other threads at any attempt to access the already-deleted object.

The fact that object threads and closures can prolong object scope means, for example, that objects assigned to local variables can exist for longer than the scope of their host variable if they have one or more methods running in other threads or if closures created from within the object still exist at the time the local variable goes out of scope.

For more information about threading, please see Threading

Since

As of Qore 0.8.10 the existence of a closure created within the object no longer prolongs the scope of the object; the closure encapsulate the object's state (along with any local variables referenced within the closure), but if the object goes out of scope while the closure still exists, then any references to the object after the object has been destroyed will cause `OBJECT-ALREADY-DELETED` exceptions to be thrown. This addresses memory and reference leaks caused by recursive references when closures encapsulating an object's scope are assigned to or accessible from members of the object.

12.3.4 Copying Objects

To explicitly generate a copy of an object, the `copy()` constructor must be called. This is a special method that exists implicitly for every class even if it is not explicitly defined (like `constructor()` and `destructor()` methods). The implicit behavior of the `copy()` constructor is to create a new object with new members that are copies of the original members (except objects are created as copies of references to the object). Then, if any `copy()` method exists, it will be executed in the new object, passing a copy of a reference to the old object as the first argument to the `copy()` method.

Note

In a class hierarchy copy() methods are called in the same order as constructor() methods.

Not all built-in classes can be copied. Classes not supporting copying will throw an exception when the `copy()` methods are called. See the documentation for each class for more information.

Chapter 13

Code Data Types

There are two types of code data types in Qore (see also Basic Data Types and Container Data Types):

- Closure Type
- Call Reference Type

13.1 Closure Type

Description:

Closures are pieces of code used as values; see Closures for more information.

Immediate Value Example:

```
string sub (int $c) { return sprintf("temp=%d", $c); }
```

Pseudo Class for Type Closure:

Qore::zzz8closurezzz9

Type Code:

Qore::NT_CLOSURE

Type Name:

```
"closure"
```

See also

Closures, code, Call Reference Type

13.2 Call Reference Type

Description:

Call references are a value type that can be used like a closures; see Call References for more information.

Immediate Value Example:

```
\function()
```

Pseudo Class for Type Call Reference:

Qore::zzz8callrefzzz9

Type Code:

Qore::NT_CALLREF

Type Name:

```
"call reference"
```

See also

Call References, code, Closure Type

Chapter 14

Data Type Declarations and Restrictions

Starting in Qore 0.8.0, it is possible to restrict variables, class members, and function and method parameters to certain data types. This allows programmers to write safer code, as many more errors can be caught at parse time that would otherwise be caught at run time. Furthermore, providing type information to the parser allows Qore to implement performance optimizations by performing lookups and resolutions once at parse time rather than every time a variable or class member is accessed at run time.

When types are declared in a parameter list, functions and methods can be overloaded as well.

The types in the following table can be used as well as any class name or `'*classname'` (i.e. an asterix followed by the class name), meaning either the given class or NOTHING (no value).

Data Type Declaration Names

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
int	Integer	Integer	Restricts values Integer values
float	Float or Integer	Float	Restricts values to Float values
number	Number, Float, or Integer	Number	Restricts values to Number values
bool	Boolean	Boolean	Restricts values to Boolean values
string	String	String	Restricts values to String values
date	Date	Date	Restricts values to Date values; values may be either absolute or relative
binary	Binary	Binary	Restricts values to Binary values
hash	Hash	Hash	Restricts values to Hash values
list	List	List	Restricts values to List values
object	Object	Object	Restricts values to Object values
<classname>	Object	Object	Restricts values to objects of the specific class given; either the class name can be given (ex: Mutex or a qualified path to the class: Qore::Thread::Mutex)
null	NULL	NULL	Restricts values to Qore's NULL type; this type has few (if any) practical applications and has been included for completeness' sake
nothing	NOTHING	NOTHING	Restricts values to Qore's NOTHING type; this type is mostly useful for declaring that a function or method returns no value
timeout	Integer, Date	Integer	Accepts Integer, Date and converts dates to an integer value representing milliseconds and returns the integer; incoming integers are assumed to represent milliseconds

softint	Integer, Float, Number, Boolean, String, NULL	Integer	Accepts Integer, Float, Boolean, String, NULL and converts non-integer values to an integer and returns the integer
softfloat	Integer, Float, Number, Boolean, String, NULL	Float	Accepts Integer, Float, Boolean, String, NULL and converts non-float values to a float and returns the new value
softnumber	Integer, Float, Number, Boolean, String, NULL	Float	Accepts Integer, Float, Boolean, String, NULL and converts non-float values to a float and returns the new value
softbool	Integer, Float, Number, Boolean, String, NULL	Boolean	Accepts Integer, Float, Boolean, String, NULL and converts non-boolean values to a boolean and returns the new value
softstring	Integer, Float, Number, Boolean, String, NULL	String	Accepts Integer, Float, Boolean, String, NULL and converts non-string values to a string and returns the new value
softdate	Integer, Float, Number, Boolean, String, Date, NULL	Date	Accepts Integer, Float, Boolean, String, Date, and NULL and converts non-date values to a date and returns the new value
softlist	all types	List	Accepts all types; NOTHING is returned as an empty list; a list is returned unchanged, and any other type is returned as the first element of a new list
data	String or Binary	same as received	Restricts input to String and Binary and returns the same type
code	Closures, Call References	same as received	Restricts values to closures and call references
reference	References	the type the reference points to	Restricts values to references to lvalues
*int	Integer, NULL, or NOTHING	Integer or NOTHING	Restricts values to Qore's Integer or NOTHING types; if NULL is passed then NOTHING is returned

*float	Float, NULL, or NOTHING	Float or NOTHING	Restricts values to Qore's Float or NOTHING types; if NULL is passed then NOTHING is returned
*bool	Boolean, NULL, or NOTHING	Boolean or NOTHING	Restricts values to Qore's Boolean or NOTHING types; if NULL is passed then NOTHING is returned
*string	String, NULL, or NOTHING	String or NOTHING	Restricts values to Qore's String or NOTHING types; if NULL is passed then NOTHING is returned
*date	Date, NULL, or NOTHING	Date or NOTHING	Restricts values to Qore's Date or NOTHING type; values may be either absolute or relative date/time values; if NULL is passed then NOTHING is returned
*binary	Binary, NULL, or NOTHING	Binary or NOTHING	Restricts values to Qore's Binary or NOTHING types; if NULL is passed then NOTHING is returned
*hash	Hash, NULL, or NOTHING	Hash or NOTHING	Restricts values to Qore's Hash or NOTHING types; if NULL is passed then NOTHING is returned
*list	List, NULL, or NOTHING	List or NOTHING	Accepts either a List or NOTHING; if NULL is passed then NOTHING is returned
*object	Object, NULL, or NOTHING	Object or NOTHING	Accepts either an Object or NOTHING; if NULL is passed then NOTHING is returned
*<classname>	Object of the given class, NULL, or NOTHING	Object of the given class or NOTHING	Restricts values to objects of the specific class given or NOTHING; either the class name can be given (ex: *Mutex or a qualified path to the class: *Qore::Thread::Mutex); if NULL is passed then NOTHING is returned

*data	String, Binary, NULL, or NOTHING	String, Binary, or NOTHING	Restricts input to String, Binary, or NOTHING and returns the same type; if NULL is passed then NOTHING is returned
*code	Closures, Call References, NULL, or NOTHING	Closures, Call References, or NOTHING	Restricts values to closures, call references and NOTHING; if NULL is passed then NOTHING is returned
*timeout	Integer, Date, NULL, or NOTHING	Integer or NOTHING	Accepts Integer, Date and converts dates to an integer value representing milliseconds and returns the integer; incoming integers are assumed to represent milliseconds. If no value or NULL is passed, then NOTHING is returned
*reference	References, NULL, or NOTHING	the type the reference points to	Restricts values to references to lvalues and NOTHING; if NULL is passed then NOTHING is returned
*softint	Integer, Float, Number, Boolean, String, NULL or NOTHING	Integer or NOTHING	Accepts Integer, Float, Number, Boolean, String, NULL and converts non-integer values to an integer and returns the integer. If no value or NULL is passed, then NOTHING is returned
*softfloat	Integer, Float, Number, Boolean, String, NULL or NOTHING	Float or NOTHING	Accepts Integer, Float, Number, Boolean, String, NULL and converts non-float values to a float and returns the new value. If no value or NULL is passed, then NOTHING is returned
*softnumber	Integer, Float, Number, Boolean, String, NULL or NOTHING	Number or NOTHING	Accepts Integer, Float, Number, Boolean, String, NULL and converts non-number values to a number and returns the new value. If no value or NULL is passed, then NOTHING is returned

*softbool	Integer, Float, Number, Boolean, String, NULL or NOTHING	Boolean or NOTHING	Accepts Integer, Float, Number, Boolean, String, NULL and converts non-boolean values to a boolean and returns the new value. If no value or NULL is passed, then NOTHING is returned
*softstring	Integer, Float, Number, Boolean, String, NULL or NOTHING	String or NOTHING	Accepts Integer, Float, Number, Boolean, String, NULL and converts non-string values to a string and returns the new value. If no value or NULL is passed, then NOTHING is returned
*softdate	Integer, Float, Number, Boolean, String, Date, NULL or NOTHING	Date or NOTHING	Accepts Integer, Float, Number, Boolean, String, Date, and NULL and converts non-date values to a date and returns the new value. If no value or NULL is passed, then NOTHING is returned
*softlist	all types	List or NOTHING	Accepts all types; NOTHING and list values are returned as the same value; NULL is returned as NOTHING, any other type is returned as the first element of a new list
any	any	same as received	Provides no restrictions on the type of value it receives and returns the same value

14.1 int

int Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
int	Integer	Integer	Restricts values to Qore's Integer type

Example

```
int sub foo(int $i) {
    return $i;
}
```

14.2 float

float Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
float	Float or Integer	Float	Restricts values to Qore's Float type

Example

```
float sub foo(float $f = M_PI) {
    return $f;
}
```

14.3 number

number Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
number	Number, Float, or Integer	Number	Restricts values to Qore's Number type

Example

```
number sub foo(number $n = 2.35e40) {
    return $n;
}
```

Since

Qore 0.8.6 introduced the number type

14.4 bool

bool Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
bool	Boolean	Boolean	Restricts values to Qore's Boolean type

Example

```
bool sub foo(bool $b) {
    return $b;
}
```

14.5 string

string Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
string	String	String	Restricts values to Qore's String type

Example

```
string sub foo(string $str = "bar") {
    return $str;
}
```

14.6 date

date Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
date	Date	Date	Restricts values to Qore's Date type; date/time values can be either absolute or relative

Example

```
date sub foo(date $d = now_us()) {
    return $d;
}
```

14.7 binary

binary Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
binary	Binary	Binary	Restricts values to Qore's Binary type

Example

```
binary sub foo(binary $b) {
    return $b;
}
```

14.8 hash

hash Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
hash	Hash	Hash	Restricts values to Qore's Hash type

Example

```
hash sub foo(hash $h = ("foo": "bar", "x": 2)) {
    return $h;
}
```

14.9 list

list Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
list	List	List	Restricts values to Qore's List type

Example

```
list sub foo(list $l = "foo", "bar")) {
    return $l;
}
```

14.10 object

object Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
object	Object	Object	Restricts values to Qore's Object type; note that any class name can also be used as a type restriction directly

Example

```
object sub foo(object $o = new Mutex()) {
    return $o;
}
```

14.11 <classname>

<classname> Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
any class name	Object of the particular class given	Object of the particular class given	Restricts values to objects of the particular class given; subclasses are also accepted

Example

```
Mutex sub foo(Qore::Thread::Mutex $m = new Mutex()) {
    return $m;
}
```

14.12 null

null Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
null	NULL	NULL	Restricts values to Qore's NULL type; this type has few (if any) practical applications and has been included for completeness' sake

Example

```
# I don't know if this type has any useful/practical applications...
my null $n = NULL;
```

14.13 nothing

nothing Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
nothing	NOTHING	NOTHING	Restricts values to Qore's NOTHING type; this type is mostly useful for declaring that a function or method returns no value

Example

```
nothing sub bar() {
    printf("foo\n");
}
```

14.14 timeout

timeout Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
`timeout`	Integer, Date	Integer	Accepts Integer, Date values and converts dates to an integer value representing milliseconds and returns the integer; incoming integers are assumed to represent milliseconds

Example

```
timeout sub foo(timeout $to = 1250ms) {
    return $to;
}
```

14.15 softint

softint Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
`softint`	Integer, Float, Number, Boolean, String, NULL	Integer	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-integer values to an integer and returns the integer

Example

```
softint sub foo(softint $i = "1000") {
    # note that "200" will be converted to an integer on return
    return $i > 500 ? "200" : $i;
}
```

14.16 softfloat

softfloat Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
`softfloat`	Integer, Float, Number, Boolean, String, NULL	Float	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-float values to a float and returns the float

Example

```
softfloat sub foo(softfloat $f = "1000") {
```

```
    # note that "200" will be converted to a float on return
    return $f > 500.0 ? "200" : $f;
}
```

14.17 softnumber

softnumber Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
softnumber	Integer, Float, Number, Boolean, String, NULL	Float	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-number values to a number and returns the number

Example

```
softnumber sub foo(softnumber $n = "1000") {
    # note that "200" will be converted to a number on return
    return $n > 500.0n ? "200" : $n;
}
```

Since

Qore 0.8.6 introduced the softnumber type

14.18 softbool

softbool Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
softbool	Integer, Float, Number, Boolean, String, NULL	Boolean	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-boolean values to a boolean and returns the boolean

Example

```
my softbool $b = "0.5";
```

14.19 softstring

softstring Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
softstring	Integer, Float, Number, Boolean, String, NULL	String	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-string values to a string and returns the string

Example

```
my softstring $str = 200;
```

14.20 softdate

softdate Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
softdate	Integer, Float, Number, Boolean, String, Date, NULL	Date	Accepts Integer, Float, Number, Boolean, String, Date, and NULL values and converts non-date values to a date and returns the date

Example

```
my softdate $d = "2001-10-10T20:00:05 +04:00";
```

14.21 softlist

softlist Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
softlist	all data types	List	Accepts all data types; NOTHING is returned as an empty list; a list is returned unchanged, and any other type is returned as the first element of a new list

Example

```
softlist sub foo(softlist $l) {
    foreach my any $element in (\$l) {
    }
    return $l;
}
```

Since

Qore 0.8.3 introduced the softlist type

14.22 data

data Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
data	String or Binary	same as received	Restricts values to String and Binary

Example

```
data sub foo(data $d) {
    return $d;
}
```

14.23 code

code Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
code	Closures, Call References	same as received	Restricts values to Closures and Call References

Example

```
sub foo(code $c) {
    $c();
}
```

Note

that also "closure" and "callref" are accepted as synonyms for "code" (they are not more specific than "code" but rather provide identical type restrictions)

14.24 reference

reference Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
reference	References	the type the reference points to	Requires a reference to an lvalue to be assigned

Example

```
sub foo(reference $f) {
    $f = 10;
}

my int $i;
foo(\$i);
```

14.25 *int

***int Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*int	Integer, NULL, or NOTHING	Integer or NOTHING	Restricts values to Integer and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*int sub foo(*int $i) {
    return $i;
}
```

14.26 *float

***float Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*float	Float, NULL, or NOTHING	Float or NOTHING	Restricts values to Float and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*float sub foo(*float $f) {
    return $f;
}
```

14.27 *number

***number Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*number	Number, NULL, or NOTHING	Number or NOTHING	Restricts values to Number and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*number sub foo(*number $n) {
    return $n;
}
```

Since

Qore 0.8.6 introduced the *number type

14.28 *bool

***bool Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*bool	Boolean, NULL, or NOTHING	Boolean or NOTHING	Restricts values to Boolean and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*bool sub foo(*bool $b) {
    return $b;
}
```

14.29 *string

***string Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*string	String, NULL, or NOTHING	String or NOTHING	Restricts values to String and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*string sub foo(*string $str) {
    return $str;
}
```

14.30 ∗date

∗date Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*date	Date, NULL, or NOTHING	Date or NOTHING	Restricts values to Date and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*date sub foo(*date $str) {
    return $str;
}
```

14.31 *binary

*binary Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*binary	Binary, NULL, or NOTHING	Binary or NOTHING	Restricts values to Binary and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*binary sub foo(*binary $b) {
    return $b;
}
```

14.32 *hash

*hash Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*hash	Hash, NULL, or NOTHING	Hash or NOTHING	Restricts values to Hash and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*hash sub foo(*hash $h) {
    return $h;
}
```

14.33 *list

*list Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*list	List or NOTHING	List or NOTHING	Restricts values to List and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*list sub foo(*list $l) {
    return $l;
}
```

14.34 *object

***list Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*list	Object, NULL, or NOTHING	Object or NOTHING	Restricts values to Object and NOTHING; if NULL is passed then NOTHING is returned

Example

```
*object sub foo(*object $obj) {
    return $obj;
}
```

14.35 *<classname>

***<classname> Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
**any class name*	Object of the particular class given, NULL, or NOTHING	Object of the particular class given or NOTHING	Restricts values to objects of the particular class given or NOTHING; subclasses are also accepted; if NULL is passed then NOTHING is returned

Example

```
sub foo(*Mutex $m) {
}
```

14.36 *date

***data Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*data	String, Binary, NULL, or NOTHING	String, Binary, or NOTHING	Restricts values to String, Binary, and NOTHING; if NULL is passed then NOTHING is returned

Example

```
sub foo(*data $d) {
}
```

14.37 *code

***code Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*code	Closures, Call References, NULL, or NOTHING	Closures, Call References, or NOTHING	Restricts values to Closures, Call References, and NOTHING; if NULL is passed then NOTHING is returned

Example

```
sub foo(*code $c) {
}
```

14.38 *timeout

*timeout Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*timeout	Integer, Date, NULL, or NOTHING	Integer or NOTHING	converts dates to an integer value representing milliseconds and returns the integer; incoming integers are assumed to represent milliseconds; also accepts NOTHING and returns NOTHING; if NULL is passed then NOTHING is returned

Example

```
sub foo(*timeout $c) {
}
```

14.39 *reference

reference Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
reference	References, NULL, or NOTHING	the type the reference points to	Requires a reference to an lvalue to be assigned or NOTHING; if NULL is passed then NOTHING is returned

Example

```
sub foo(*reference $f) {
    $f = 10;
}

my int $i;
foo(\$i);
```

14.40 *softint

*softint Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softint	Integer, Float, Number, Boolean, String, NULL, NOTHING	Integer or NOTHING	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-integer values to an integer and returns the integer; also accepts NOTHING and NULL and returns NOTHING

Example

```
sub foo(*softint $i) {
}
```

14.41 *softfloat

***softfloat Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softfloat	Integer, Float, Number, Boolean, String, NULL, NOTHING	Float or NOTHING	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-float values to a float and returns the float; also accepts NOTHING and NULL and returns NOTHING

Example

```
sub foo(*softfloat $f) {
}
```

14.42 *softnumber

***softnumber Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softnumber	Integer, Float, Number, Boolean, String, NULL, NOTHING	Number or NOTHING	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-number values to a number and returns the number; also accepts NOTHING and NULL and returns NOTHING

Example

```
sub foo(*softnumber $n) {
}
```

Since

Qore 0.8.6 introduced the *softnumber type

14.43 *softbool

***softbool Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softbool	Integer, Float, Number, Boolean, String, NULL, NOTHING	Boolean or NOTHING	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-boolean values to a boolean and returns the boolean; also accepts NOTHING and NULL and returns NOTHING

Example

```
sub foo(*softbool $b) {
}
```

14.44 *softstring

***softstring Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softstring	Integer, Float, Number, Boolean, String, NULL, NOTHING	String or NOTHING	Accepts Integer, Float, Number, Boolean, String, and NULL values and converts non-string values to a string and returns the string; also accepts NOTHING and returns NOTHING

Example

```
sub foo(*softstring $str) {
}
```

14.45 *softdate

***softdate Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softdate	Integer, Float, Number, Boolean, String, Date, NULL, NOTHING	String or NOTHING	Accepts Integer, Float, Number, Boolean, String, Date, and NULL values and converts non-date values to a date and returns the date; also accepts NOTHING and NULL and returns NOTHING

Example

```
sub foo(*softdate $d) {
}
```

14.46 *softlist

***softlist Type Restriction**

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
*softlist	all data types	List or NOTHING	Accepts all data types; NOTHING and NULL are returned as NOTHING; a list is returned unchanged, and any other type is returned as the first element of a new list

Example

```
my *softlist $d = $v;
```

Since

Qore 0.8.3 introduced the *softlist type

14.47 any

any Type Restriction

Name	Accepts Qore Type(s)	Returns Qore Type(s)	Description
any	all data types	all data types	Accepts all data types and returns the same data type

Example

```
my any $v = $bar;
```

Note

Complex types (hash of lists, reference to string, etc) are currently not possible to declare but support may be added in future versions of Qore for this.

Chapter 15

References

References allow for addressing other lvalues, including complex lvalue expressions, with an alias called a reference. References to lvalues are defined by placing a "\" character in front of an expression that gives an lvalue, such as in the following examples:

```
my reference $r1 = \$a;
my reference $r2 = \$recs[$rn].order[$on];
```

References are especially convenient when aliasing an internal part of a complex data structure. Consider the following example (a list of records, where each record is a hash with a list of orders under the "order" key - in the following example, we want to set the last record's last order's "numberofitems" key to be equal to the number of elements under the order's "items" key):

```
$recs[$recs.size() - 1].order[$recs[$recs.size() - 1].order.size() - 1].numberofitems = $recs[$recs.size()
      - 1].order[$recs[$recs.size() - 1].order.size() - 1].items.size();
```

In the above code, using references could greatly simplfy the readability of the code:

```
my reference $lastrec = \$recs[$recs.size() - 1];
my reference $lastorder = \$lastrec.order[$lastrec.order.size() - 1];
$lastorder.numberofitems = $lastorder.items.size();
```

Note

local variables that are referenced are automatically converted to a special type of local variable that is protected by a mutual-exclusion thread lock so that the reference can be safely used in background threads. The lvalue represented by the local variable will exist beyond its local scope; as long as the reference to the lvalue exists, the local variable's lvalue will exist and remain valid.

Since

Qore 0.8.5 universal lvalue references are supported; previously references could only be used with arguments to a function or method call or the like. As of Qore 0.8.5+, references can be used anywhere in Qore in any expression, even with references to local variables in background expressions and in closure argument lists.

Chapter 16

Overloading

Functions and methods can be overloaded if parameter types are declared as in the following example:

```
int sub example(int $i) {
    printf("i=%d\n", $i);
    return $i + 1;
}

string sub example(string $str) {
    printf("str=%s\n", $str);
    return $str + "foo";
}
```

In this case, the first version (`example(int)`) will be executed if called with an integer argument, and the second (`example(string)`) if called with a string argument.

Class methods may also be overloaded, but note that destructor(), copy(), methodGate(), memberGate(), and memberNotification() methods may not be overloaded (see Classes for more information).

Chapter 17

Time Zone Handling

Qore assumes a default time zone for all programs when it starts up. The rules for determining the default time zone are similar to those for the C library in most UNIX or UNIX-like operating systems.

17.1 UNIX Time Zone Handling

If the `TZ` environment variable is defined, then the contents of that variable are used to find a zoneinfo file that contains the time zone definition. If this file cannot be found, then the default time zone will default to `UTC`.

If the `TZ` environment variable is not defined or is empty, then the Qore library tries to find the default zoneinfo definition file (normally `/etc/localtime`). If found, this file is read in and provides the information about the local time zone. If not found, the default time zone will default to `UTC`.

When a zoneinfo file is found, information about local time zone names and daylight savings time is available for times tagged with that time zone.

Note that posix-style time zone rules are not understood if assigned to the `TZ` environment variable, only file names to a zoneinfo file can be processed at the moment. Furthermore if the zoneinfo file contains leap second information, it is currently ignored.

17.2 Windows Time Zone Handling

Time zone information is read in from binary time zone data in the Windows registry under:

- `HKEY_LOCAL_MACHINE SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones`

Time zone region names must correspond to registry keys under the above key or the time zone information will not be loaded.

The region name reported by Qore::TimeZone::region() will be the display name for the time zone, as found in the registry.

For example:

```
O:\bin>qore -ne "TimeZone tz('Central Europe Standard Time'); printf(\"%s\n\", tz.region());"
(GMT+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
```

This differs from UNIX where the region name passed to the TimeZone constructor is the same region name that appears in the Qore::TimeZone::region() output; this is because Windows uses `"Standard Time"` in the key name in the registry, even though the underlying zone definition is for both standard and daylight savings time.

Note

Please be aware that the region names may differ in localized versions of Windows

17.3 More Time Zone Information and Examples

See the Qore::TimeZone class for information about retrieving, setting, and querying time zone information; see Date and Time Functions for a list of functions related to date/time processing.

Here are some examples using Qore's '-X' option for evaluating an expression and displaying the result immediately:

```
prompt% TZ=America/Chicago qore -X 'now_us()'
2010-05-11 06:14:28.845857 Tue -05:00 (CDT)
prompt% TZ=Europe/Rome qore -X 'now_us()'
2010-05-11 13:14:35.070568 Tue +02:00 (CEST)
prompt% TZ=Australia/Sydney qore -X 'now_us()'
2010-05-11 21:14:45.422222 Tue +10:00 (EST)
prompt% TZ=Asia/Tokyo qore -X 'now_us()'
2010-05-11 20:14:59.609249 Tue +09:00 (CJT)
```

Chapter 18

Strings and Character Encoding

18.1 Overview

The Qore language is character-encoding aware. All strings are assumed to have the default character encoding, unless the program explicitly specified another encoding for certain objects and operations. Every Qore string has a character encoding ID attached to it, so, when another encoding is required, the Qore language will attempt to do an encoding translation.

Qore uses the operating system's `iconv` library functions to perform any encoding conversions.

Qore supports character encodings that are backwards compatible with 7-bit `ASCII`. This includes all `ISO-8859-*` character encodings, `UTF-8`, `KOIR-8`, `KOIU-8`, and `KOI7`, among others (see the table below: Known Character Encodings).

However, mutibyte character encodings are currently only properly supported for `UTF-8`. For `UTF-8` strings, the length(), index(), rindex(), substr(), reverse(), the splice operator, print formatting (regarding field lengths) functions and methods taking format strings, and regular expression operators and functions, all work with character offsets, which may be different than byte offsets. For all character encodings other than `UTF-8`, a 1 byte=1 character relationship is assumed.

Qore will accept any encoding name given to it, even if it is not a known encoding name or alias. In this case, Qore will tag the strings with this encoding, and pass this user-defined encoding name to the `iconv` library when encodings must be converted. This allows programmers to use encodings known by the system's `iconv` library, but unknown to Qore. In this case, Qore will assume that the strings are backwards compatible with `ASCII`, meaning that that one character is represented by one byte and that the strings are null-terminated.

Note that when Qore matches an encoding name to a code or alias in the following table, the comparison is not case-sensitive.

18.2 Character Encodings Known to Qore

Code	Aliases	Description
`ISO-8859-1`	`ISO88591`, `ISO8859-1`, `ISO-88591`, `ISO8859P1`, `ISO81`, `LATIN1`, `LATIN-1`	latin-1, Western European character set
`ISO-8859-2`	`ISO88592`, `ISO8859-2`, `ISO-88592`, `ISO8859P2`, `ISO82`, `LATIN2`, `LATIN-2`	latin-2, Central European character set

ISO-8859-3	ISO88593, ISO8859-3, ISO-88593, ISO8859P3, ISO83, LATIN3, LATIN-3	latin-3, Southern European character set
ISO-8859-4	ISO88594, ISO8859-4, ISO-88594, ISO8859P4, ISO84, LATIN4, LATIN-4	latin-4, Northern European character set
ISO-8859-5	ISO88595, ISO8859-5, ISO-88595, ISO8859P5, ISO85	Cyrillic character set
ISO-8859-6	ISO88596, ISO8859-6, ISO-88596, ISO8859P6, ISO86	Arabic character set
ISO-8859-7	ISO88597, ISO8859-7, ISO-88597, ISO8859P7, ISO87	Greek character set
ISO-8859-8	ISO88598, ISO8859-8, ISO-88598, ISO8859P8, ISO88	Hebrew character set
ISO-8859-9	ISO88599, ISO8859-9, ISO-88599, ISO8859P9, ISO89, LATIN5, LATIN-5	latin-5, Turkish character set
ISO-8859-10	ISO885910, ISO8859-10, ISO-885910, ISO8859P10, ISO810, LATIN6, LATIN-6	latin-6, Nordic character set
ISO-8859-11	ISO885911, ISO8859-11, ISO-885911, ISO8859P11, ISO811	Thai character set
ISO-8859-13	ISO885913, ISO8859-13, ISO-885913, ISO8859P13, ISO813, LATIN7, LATIN-7	latin-7, Baltic rim character set
ISO-8859-14	ISO885914, ISO8859-14, ISO-885914, ISO8859P14, ISO814, LATIN8, LATIN-8	latin-8, Celtic character set
ISO-8859-15	ISO885915, ISO8859-15, ISO-885915, ISO8859P15, ISO815, LATIN9, LATIN-9	latin-9, Western European with euro symbol
ISO-8859-16	ISO885916, ISO8859-16, ISO-885916, ISO8859P16, ISO816, LATIN10, LATIN-10	latin-10, Southeast European character set
KOI7	n/a	Russian: Kod Obmena Informatsiey, 7 bit characters
KOI8-R	KOI8R	Russian: Kod Obmena Informatsiey, 8 bit
KOI8-U	KOI8U	Ukrainian: Kod Obmena Informatsiey, 8 bit
US-ASCII	ASCII, USASCII	7-bit ASCII character set
UTF-8	UTF8	variable-width universal character set

18.3 Default Character Encoding

The default character encoding for Qore is determined by environment variables.

First, the QORE_CHARSET environment variable is checked. If it is set, then this character encoding will be the default character encoding for the process. If not, then the LANG environment variable is checked. If a character encoding is specified in the LANG environment variable, then it will be used as the default character encoding. Otherwise, if no character encoding can be derived from the environment, UTF-8 is assumed.

Character encodings are automatically converted by the Qore language when necessary. Encoding conversion errors will cause a Qore exception to be thrown. The character encoding conversions supported by Qore depend on the operating system's `iconv` library function.

Note

The get_default_encoding() function will return the default encoding for the Qore process.

18.4 Character Encoding Usage Examples

The following is a non-exhaustive list of examples in Qore where character encoding processing is performed.

Character encodings can be explicitly performed with the convert_encoding() function, and the encoding attached to a string can be checked with the get_encoding() function. If you have a string with incorrect encoding and want to change the encoding tag of the string (without changing the actual bytes of the string), use the force_encoding() function.

get_default_encoding() returns the default encoding for the Qore process.

The Qore::SQL::Datasource, Qore::SQL::DatasourcePool, and Qore::SQL::SQLStatement classes will translate character encodings to the encoding required by the database if necessary as well (this is actually the responsibility of the DBI driver for the database in question).

The Qore::File and Qore::Socket classes translate character encodings to the encoding specified for the object if necessary, as well as tagging strings received or read with the object's encoding.

The Qore::HTTPClient class will translate character encodings to the encoding specified for the object if necessary, as well as tag strings received with the object's encoding. Additionally, if an HTTP server response specifies a specific encoding to use, the encoding of strings read from the server will be automatically set to this encoding as well.

Chapter 19

Expressions

An expression can be any of the following (note that expressions are also recursively defined):

Expressions

Type	Description	Examples

An immediate value	Qore values that can be expressed directly (see Basic Data Types and Container Data Types for more information)	`True`
		`1.2`
		`"a string"`
		`2005-10-27`
		`NULL`
		`NOTHING`
		`("key" : $val)`

A variable reference	see also %allow-bare-refs	`$var`
A variable declaration	Variable Declarations and Lexical Scope, see also %assume-local	`my int $var`
An in-class object member reference	References to members of an object from within the class see see also allow-bare-refs "%allow-bare-refs"	`$.member`
An lvalue assignment	Assigns a value to a lvalue (see Assignment Operator (=))	`$var = 1` `($a, $b, $c, $date) = (1, "two", 3.3, 2005-10-28)`
A function call	Qore function calls (see Functions)	`calculate($var1, $var2, "string", 4)`
A method call	Qore object method calls (see Object Method Calls) see also %allow-bare-refs	`$object.method("argument")`

An in-class method call	Qore in-class object method calls (see Object Method Calls) see also %allow-bare-refs	`$.method("argument")`
A static method call	Qore static method calls (see static_methods)	`ClassName::static_method("argument")`

Expressions with operators	Use of Qore operators	`1 + 2`
		`$a || $b`
		`background my_function()`

An expression in parentheses	Use of parentheses for clarity or to specify evaluation precedence	`(3 * (2 + $a))`
A find expression	Finds a value or values in a hash of lists, such as returned by the Qore::SQL::Datasource::select() or Qore::SQL::SQLStatement↩ ::fetchColumns() method	`find %name, %id in $data where (%name =~ /Smith/)`
A context reference (`name`)	A contextual reference to the value of a key of the current row being iterated by a context, summarize, subcontext statement, or a find expression	`%name`
A context row reference (`%%`)	A contextual reference to the current row being iterated by a context, summarize, subcontext statement, or a find expression; this expression returns a hash of the current row	`%%`
A call reference	A reference to a function or object method call (similar to a function pointer in C or C++). Function references are resolved in the second phase of parsing (commit phase), while object method references are resolved at run-time	`\function_call()` `\$object_expression.method_name()`
A closure	An anonymous function used a value; technically a closure must have at least one bound variable, but in Qore a closure is any function used as a value, whether or not it encloses local variables from the scope in which it was created or not	`string sub (string $a) { return $a + $b; }`

A call reference call	An expression executing a call reference or closure	`$my_closure($arg1, $arg2)`
An implicit argument reference (`$1`)	References an implicit argument	`$1`
A reference to the entire implicit argument list (`$$`)	References the implicit argument list	`$$`
An implicit index reference	Gives the list index position when implicitly iterating a list	`$#`

19.1 Static Method Calls

Synopsis

Calls to static class methods are made by giving the class name followed by two colons and then the method name. The method name must be implemented and accessible (i.e. not private and accessed outside the class) somewhere within the class hierarchy and must be static or a parse exception will occur.

Syntax

class_name::method_name (*[argument_expressions...]*)

Description

class_name
The name of the class implementing the static method.

method_name
The name of the static method to call.

[argument_expressions...]
Expressions passing arguments to the static method.

Example

```
TimeZone::setRegion("Europe/Prague");
```

19.2 Find Expressions

Synopsis

The find expression can be used to quickly find data in a hash of lists (such as a query result set returned by the Qore::SQL::Datasource::select() or Qore::SQL::SQLStatement::fetchColumns() methods). The find expression will loop through a data structure, and for each element in the structure where the `where` expression is True, it will evaluate and return a result expression.

If the *where_expression* only is True for one element in the list, it will return the result of evaluating the result expression directly, otherwise if the *where_expression* is True more than once, then a list of the results of evaluting the result expression for each element is returned.

In each expression in the find expression, column values can be referred to by preceding the name with a "`%`" character (as with context statements).

Syntax

find *result_expression* **in** *data_expression* **where** (*where_expression)*

Description

result_expression
This expression will be evaluated and returned when the *where_expression* evaluates to True.

data_expression
This expression must evaluate to a hash of lists, so that the internal context can be set up for the find loop.

where_expression
This expression will be evaluated for each row in the *data_expression*. Each time it evaluates to True, the *result_expression* will be evaulated and used for the return value for the find expression.

Example

```
$rlist = find %fname, %id in $data where (%lname =~ /^Smith/);
```

See also

- context
- summarize
- subcontext

19.3 Call References

Synopsis

References to functions or object methods are called call references. A call reference can be used like a function pointer; a call reference is a Qore data type that can be returned by functions or methods or assigned to variables.

Note that the empty parentheses after the call are required to identify the expression as a call reference.

Syntax

\ *function_name* ()
\ *class* **::** *static_method* ()
\ *object.method* ()

Description

\ *function_name* ()
This makes a call reference to a function. Call references to functions are resolved at parse time; if the function does not exist a parse exception will be thrown.

\ *class* **::** *static_method* ()
This makes a call reference to a static method. Call references to static methods are resolved at parse time; if the static method does not exist a parse exception will be thrown.

\ *object* **.** *method* ()

- *object:* can be any valid Qore expression that evaluates to an object
- *method:* must be an unquoted string (see example below) and must represent a valid method name of the object's class.

This makes a call reference to an object method call, binding the object and the method in the call reference. Call references to object methods are executed and resolved at run time; if the object expression does not evaluate to an object at run-time, an `OBJECT-METHOD-REFERENCE-ERROR` exception will be thrown. If the method does not exist, a `METHOD-DOES-NOT-EXIST` run-time exception will be thrown.

When called, a call reference to an object method will be executed in the context of the object originally referenced. Object method call references do not prolong the lifetime of an object; if the object is deleted (for example, by going out of scope), then if called the call reference will cause an `OBJECT-ALREADY-DELETED` exception to be thrown.

Example

```
my code $c = \printf();
my code $c = \MyClass::method();
my code $c = \$obj.method();
```

Note

- The backslash at the beginning and the empty parentheses at the end; these are required when specifying a call reference.
- call reference is a code data type; see Call Reference Type for more information

19.4 Closures

Synopsis

A closure is an anonymous function used as a value. Closures can be returned from functions or methods, assigned to variables, or passed as arguments to other functions.

Syntax

[return_type] **sub** `(` `[` *[type] variable1, ...* `]` `)` `{` *[code]* `}`

or the alternate (deprecated) syntax with the `returns` keyword after the parameters:

sub `(` `[` *[type] variable1, ...* `]` `)` **returns** *return_type* `{` *[code]* `}`

Description

Closures encapsulate the state and value of local variables of the outer code block referenced from within the closure when the closure is created. Whenever local variables are bound within a closure, these variables are subject to concurrent thread access protection (locking) just as with global variables, in order to allow closures to be used in any context without restriction and to preseve thread-safety regarding bound local variables.

Note that returning a closure from within an object method encapsulates the state of the object as well (it's legal to refer to `$self` and `$.`*member* from within closures created from objects) and additionally prolongs the scope of the object for the lifetime of the closure.

Note that parameter and return types are required when the Qore::PO_REQUIRE_TYPES or Qore::PO_REQUIRE_PROTOTYPES parse options are set.

Example

```
# if $b is a local variable in the function where the closure is created
# then $b will be bound to the closure when the closure is created
my code $closure = int sub (int $a) { return $a + $b; };
```

Note

closure is a code data type; see Closure Type for more information

19.5 Implicit Argument References

Synopsis

Implicit arguments are arguments not captured by parameter variables as well as automatic arguments in list-processing operator expressions. A special syntax to reference these arguments is documented here.

Syntax

`$`*int* (for a single implicit argument; *int* is the argument number, where `1` is the first argument)
`$$` (for the entire implicit argument list)

Description

Implicit arguments can be directly referenced using the dollar sign (`$`) and either a number from `1` onwards (giving the position in the argument list, where `1` is the first element) or a double dollar sign (`$$`) giving the entire implicit argument list.

For unassigned arguments to functions or methods, this syntax supplements the automatic $argv variable holding all function arguments not assigned to parameter variables.

This syntax is particularly useful when writing expressions for the map, foldl, foldr, and select operators, where implicit argument references are the only way the operator expressions can reference the current list values that are populated as implicit arguments as the operators traverse the list.

Example

```
# extract a list of even numbers from a list
my list $l = select $list, !($1 % 2);
```

19.6 Implicit Index

Synopsis

The current list index position when implicitly iterating through lists can be referenced using the implicit index reference characters: `$#`.

Syntax

`$#`

Description

The implicit index reference expression (`$#`) can be used whenever a list is iterated implicitly, such as with foreach statements and the map, foldl, foldr, and select operators.

Example

```
# create a list of indexes with negative values
my list $l = map $#, $list, ($1 < 0);
```

Chapter 20

Operators

The following table lists all Qore operators in order of precedence, starting with the highest precedence. The lower the precedence number, the higher the precedence, therefore the operators with precedence level 1 ("`{}`", "`[]`", "`.`") have the highest precedence of all Qore operators. The precedence levels in Qore are roughly equal to the precedence levels of C language operators. To explicitly specify the precedence for expression evaluation, use parentheses `()`.

Operators

Operator	Prec.	Description	Example
“	1	backquote/backtick operator	`` `ls -l` ``
{}	1	hash element or object member expression dereference operator	`$hash{"na" + "me"}`
.	1	hash element or object member literal dereference operator	`$hash.name` `$obj.method()`
[]	1	list element, string, and binary dereference operator	`$list[1]`
++	2	pre-increment operator	`++$a`
++	2	post-increment operator	`$a++`
--	2	pre-decrement operator	`--$a`
--	2	post-decrement operator	`$a--`
new	3	class instantiation/new object operator	`new Socket()`

background	3	background/thread creation operator	`background mainThread()`
delete	3	delete operator	`delete $var`
remove	3	remove operator	`remove $var`
cast<>()	3	cast<>() operator	`cast<SubClass>($var)`
!	4	logical negation operator	`if (!($a > 10)) {}`
~	5	binary not/bit inversion operator	`$var = ~$var`
- (unary minus)	6	unary minus operator	`$var = -$var`
shift	7	shift list element operator	`shift $list`
pop	7	pop list element operator	`pop $list`
chomp	7	chomp end-of-line character operator	`chomp $string`
trim	7	trim characters operator	`trim $string`
elements	8	number of elements operator (list, hash, string, binary)	`elements $list`
keys	8	hash key list operator	`keys $hash`
*	9	multiplication operator	`$var = $a * 10`
/	9	division operator	`$var = $a / 10`
%	10	modula operator	`$var = $a % 10`

+	11	plus operator: string, binary, list, and hash concatenation, integer and float addition	`$a + 10`
			`"hello" + "there"`
			`$list + "new value"`
			`$hash + ( "newkey" : 100 )`

-	11	minus operator (arithmetic subtraction, hash key removal)	`$a - 10`
>>	12	bitwise shift right operator	`0xff00 >> 8`
<<	12	bitwise shift left operator	`0xff00 << 8`
exists	13	exists value operator	`exists $var`
instanceof	13	instanceof class operator	`instanceof Qore::Mutex`
<	14	Logical less than operator	`$a < 10`
>	14	Logical greater than operator	`$a > 10`
==	14	Logical equality operator	`$a == 10`
!=	14	logical inequality operator	`$a != 10`
<=	14	Logical less then or equals operator	`$a <= 10`
>=	14	logical greater than or equals operator	`$a >= 10`
<=>	14	logical comparison operator	`$a <=> $b`
===	14	absolute logical equality operator	`$a === 10`
!==	14	absolute logical inequality operator	`$a !== 10`
=~ //	14	regular expression match operator	`$a =~ /text/`
!~ //	14	regular expression no match operator	`$a !~ /text/`
=~ s///	14	regular expression substitution operator	`$a =~ s/text/text/`

=~ x//	14	regular expression pattern extraction operator	`$a =~ x/(\w+):(\w+)/`
=~ tr	14	transliteration operator	`$a =~ tr/a-z/A-Z/`
&	15	bitwise/binary AND operator	`$a & 0xff`
|	15	bitwise/binary OR operator	`$a | 0xff`
^	15	bitwise/binary XOR operator	`$a ^ 0xff`
&&	16	logical AND operator	`($a = 1) && ($b < 10)`
||	16	logical OR operator	`($a = 1) \|\| ($b < 10)`
? :	17	conditional operator	`$a == 2 ? "yes" : "no"`
,	18	comma operator	`1, 2, 3, 4, 5`
unshift	19	unshift list element operator	`unshift $list, $val`
push	19	push list element operator	`push $list, $val`
splice	19	splice list or string operator	`splice $list, 2, 2, (1, 2, 3)`
extract	19	extract list or string operator	`my $sublist = extract $list, 2, 2, (1, 2, 3`
map	19	map operator	`map $closure($1), $list`
foldl	19	fold left to right operator	`foldl $closure($1 - $2), $list`
foldr	19	fold right to left operator	`foldr $closure($1 - $2), $list`
select	19	select elements from list operator	`select $list, $1 > 1`

=	20	assignment operator	`$var = 1`
+=	21	plus-equals (add-to) operator	`$var += 5`
-=	21	minus-equals (subtract-from) operator	`$var -= 5`
&=	21	and-equals operator	`$var &= 0x2000`
|=	21	or-equals operator	`$var |= 0x2000`
%=	21	modula-equals operator	`$var %= 100`
*=	21	multiply-equals operator	`$var *= 10`
/=	21	divide-equals operator	`$var /= 10`
^=	21	xor-equals operator	`$var ^= 0x2000`
<<=	21	shift-left-equals operator	`$var <<= 0x2000`
>>=	21	shift-right-equals operator	`$var >>= 0x2000`

Note

All Qore operators perform thread-atomic actions with respect to the immediate arguments of the operator. If the operators are used in a complex expression, the entire expression is not thread-atomic unless explicit user-level locking is used. For example: `$a += 5` is a thread-atomic action, but `$a += $b-` is not atomic, but rather made up of two atomic actions.

When an operator taking more than one argument is used with arguments of different data types, Qore automatically converts one or both data types to a data type supported by the operator in order to evaluate the result, according to the precedence lists given for each operator. That is; when an operator operates on mixed types, the types listed first in the following sections have precedence over types listed farther down in the lists. The result type will always be equal to the final operation type after any conversions due to type precedence per operator. If no type of either argument matches a supported data type for the operator, both types will be converted to the highest precedence data type for the operator and then the operator will evaluate the result. For explicit type conversion, please see the boolean(), string(), date(), int(), float(), etc functions.

20.1 Backquote Operator (“)

Synopsis

Executes the shell command in a separate process and returns the stdout as a string. To perform the same action using a Qore expression, see the backquote() function.

Syntax

`` `shell_command` ``

Return Type

string

Example

```
my string $dir = `ls -l`
```

Arguments Processed by “

Argument	Returns	Processing
unquoted string *shell_command*	string	The shell command will be executed and the `stdout` is returned as a string

Exceptions

BACKQUOTE-ERROR	An error occurred in fork() or creating the output pipe

20.2 Hash Element or Object Member Expression Dereference Operator ({})

Synopsis

Retrieves the value of hash key or object member by evaulating an expression.

Syntax

container_expression { *expression* }

Return Type

any

Example

```
printf("%s\n", $hash{getName()});
```

Arguments Processed by {}

Argument	Processing
container_expression	This expression must evaluate to a hash or an object; if not, the operator returns no value (NOTHING)
expression	- list: if the expression evaluates to a list, then a slice of the hash or object is returned as a hash containing keys given in the list that are present in the hash or object. If the key as given in the list (converted to a string if necessary) is not present in the hash or object, then it is also not present in the hash returned; of none of the given keys exist in the hash, an empty hash is returned. - anything other than list: the expression is converted to a string (if necessary); the value of the hash key corresponding to this string will be returned; if the key or member does not exist, then no value is returned

Exceptions

PRIVATE-MEMBER	Attempt to access a private member outside the class

20.3 Hash Element or Object Member Literal Dereference Operator (.)

Synopsis

Retrieves the value of a hash key or object member using a literal identifier or an expression.

Syntax

hash_or_object_expression **.** *identifier*
object_expression **.** *method_name([args ...])*
hash_or_object_expression **.** *expression*

Return Type

any

Example

```
printf("%s\n", $hash.name);

$obj.method("argument");
```

Arguments Processed by . (hash key or object member literal dereference)

Argument	Processing
hash_or_object_expression	This expression must evaluate to a hash or an object; if not, then the operator returns no value
identifier	An unquoted string taken as the literal name of the hash key or object member. If no such key exists, then no value is returned. In order to use hash keys that are not valid Qore identifiers, please use the {} operator. If the member is a private member and access is made outside the class, a run-time exception will be thrown. Also note that constants or static class member names will not be resolved, in this case the string given is used as the literal name of the hash key or object member

Arguments Processed by . (object method call)

Argument	Processing
object_expression	The *object_expression* must evaluate to an object or a run-time exception is thrown
method_name ([*args*])	If the method does not exist in the class a run-time exception is thrown. Otherwise the method is called with any optional arguments given and the return value of the method is returned.

Arguments Processed by . (hash key or object member expression dereference)

Argument	Processing
hash_or_object_expression	This expression must evaluate to a hash or an object; if not, then the operator returns no value

expression	- list: if the expression evaluates to a list, then a slice of the hash or object is returned as a hash containing keys given in the list that are present in the hash or object. If the key as given in the list (converted to a string if necessary) is not present in the hash or object, then it is also not present in the hash returned; of none of the given keys exist in the hash, an empty hash is returned. - anything other than list: the expression is converted to a string (if necessary); the value of the hash key corresponding to this string will be returned; if the key or member does not exist, then no value is returned

Exceptions

PRIVATE-MEMBER	Attempt to access a private member outside the class
METHOD-DOES-NOT-EX↩ IST	Attempt to access a method not defined for this class
METHOD-IS-PRIVATE	Attempt to access a private method from outside the class
BASE-CLASS-IS-PRIVATE	Attempt to access a method of a privately-inherited base class from outside the class
OBJECT-METHOD-EVAL-ON-NON-OBJECT	Attempt to execute a method on a non-object

20.4 List, String, and Binary Dereference Operator ([ ])

Synopsis

Retrieves the value of a list element, the given character of a string, or the integer value of a byte for a binary object. If the index value is not valid for the argument, NOTHING is returned. Note that this operator only works as a list dereferencing operator in lvalue expressions; you cannot assign a character or a byte value to strings or binaries using this operator.

Syntax

list_expression [*offset_expression]* *string_expression* [*offset_expression]* *binary_expression* [*offset_↩ expression]*

Return Type

any

Example

```
printf("%s\n", $list[2]);

printf("%s\n", $str[2]);

printf("0x%x\n", $binary[2]);
```

Arguments Processed by []

Argument	Processing
list_expression	If the expression evaluates to a list, then the *offset_expression* will be used to return the given element from the list

string_expression	If the expression evaluates to a string, then the *offset_expression* will be used to return the given character from the list; note that multi-byte characters with UTF-8 are properly respected with this operator
binary_expression	If the expression evaluates to a binary, then the *offset_expression* will be used to return the integer value of the byte given from the binary object
offset_expression	The expression is evaluated and converted to an integer if necessary. Then the value of the given element given is returned according to the type of the first expression (as listed above; elements start at position 0)

This operator does not throw any exceptions; if the first expression does not evaluate to either a list, string, or binary, then no value (NOTHING) is returned.

20.5 Pre-Increment Operator (++)

Synopsis

Increments an lvalue and returns the incremented value; only works on integer and floating-point values.

Syntax

++*lvalue*

Return Type

int or float

Example

```
++$i;
```

Arguments Processed by ++ (pre-increement)

Argument	Processing
Float	increments *lvalue* and returns the result
Integer (or any other type)	First converts the value of *lvalue* to an integer if necessary, then increments *lvalue* and returns the result

This operator does not throw any exceptions.

20.6 Integer Post-Increment Operator (++)

Synopsis

Increments an integer or floating-point lvalue and returns the value before the increment; if the lvalue is neither a floating-point value or an integer, it will be converted to an integer

Syntax

lvalue++

Return Type

any

Example

```
$i++;
```

Arguments Processed by ++ (post-increment)

Argument	Processing
Float	saves the value of the lvalue as the result, then increments *lvalue*, then returns the saved original value of *lvalue*
Integer (or any other type)	saves the value of the lvalue as the result, then converts the value of *lvalue* to an integer if necessary and increments it, then returns the saved original value of *lvalue*

This operator does not throw any exceptions.

20.7 Integer Pre-Decrement Operator (--)

Synopsis

Decrements an lvalue and returns the incremented value; only works on integer and floating-point values.

Syntax

−lvalue

Return Type

int or float

Example

```
--$i;
```

Arguments Processed by – (pre-decrement)

Argument	Processing
Float	increments *lvalue* and returns the result
Integer (or any other type)	First converts the value of *lvalue* to an integer if necessary, then increments *lvalue* and returns the result

This operator does not throw any exceptions.

20.8 Integer Post-Decrement Operator (--)

Synopsis

Decrements an integer or floating-point lvalue and returns the value before the decrement; if the lvalue is neither a floating-point value or an integer, it will be converted to an integer

Syntax

lvalue−

Return Type

any

Example

$i–;

Arguments Processed by – (post-decrement)

Argument	Processing
Float	saves the value of the lvalue as the result, then decrements *lvalue*, then returns the saved original value of *lvalue*
Integer (or any other type)	saves the value of the lvalue as the result, then converts the value of *lvalue* to an integer if necessary and decrements it, then returns the saved original value of *lvalue*

This operator does not throw any exceptions.

20.9 New Object Operator (new)

Synopsis

Creates an instance of a class by running the class' constructor on the new class (if any exists) and returns the new object.

Note that if possible it is normally better to declare an object with its type and use the abbreviated form to construct the object as follows:

```
my Mutex $m();
```

This provides type information to the parser which allows more errors to be caught at parse time (instead of at run time), and furthermore allows Qore to improve performance by performing more work once at parse time rather than for every time the object is accessed at run time (for example, method and variant resolution), and normally requires less typing.

Syntax

new *class_identifier*(*constructor_args ...*)

Return Type

an object of the specific class given

Example

```
my Mutex $obj = new Qore::Mutex();
```

Arguments Processed by new

Argument	Processing
class_identifier	The class_identifier must be an existing class name; if so, the operator instantiates an object of this class, executes the constructor for the class (if any exists, along with any base class constructors, if applicable) on the new object, and returns the object (for constructor execution order in an inherited class, see Class Inheritance). If an exception is thrown in the constructor, the object is deleted immediately.

Note

See class documentation for possible exceptions

20.10 Background Operator (background)

Synopsis

Start a background thread and return the TID (thread ID).

Note

- expressions that have no effect cannot be used with the background operator (a parse exception will be thrown)
- it is illegal to make changes to a local variable anywhere in a background expression (a parse exception will be thrown)
- local variables and find expressions are evaluated before the new thread is started and the result of the evaluation is used in the expression in the new thread
- it is not possible to pass local variables by reference anywhere in a background expression (a parse exception will be thrown)

Syntax

background *expression*

Return Type

int

Example

```
my int $tid = background startThread();
```

Arguments Processed by background

Argument	Processing
expression	The *expression* given as an argument will be executed in a new thread. The TID of the new thread will be returned as the return value of the operator

Exceptions

THREAD-CREATION-FAILURE	If the thread table is full or if the operating system returns an error while starting the thread, this exception is thrown

20.11 Delete Operator (delete)

Synopsis

The delete operator deletes the contents of an lvalue. If the delete operator is called on an object, the object will be destroyed unconditionally. The delete operator does not return any value.

When called on a hash key, the key is removed from the hash entirely; when called on a list element, the element is assigned NOTHING (i.e. the list size does not change).

The delete operator will delete multiple keys from a hash (i.e. delete a slice from a hash) when called on a hash dereferenced by a list of strings, giving the keys to delete (see example below).

In the case the delete operator operates on an object, any exception can be thrown that is thrown by the class' destructor.

For a similar operator that returns the value that is removed from the data structure, and does not delete objects, see the remove operator.

Syntax

delete *lvalue_expression*

Return Type

Does not return any value

Example

```
# delete a single key from a hash
delete $value;
# delete multiple values from a hash
delete $h.("a", "b", "c");
```

This operator does not throw any exceptions, however exceptions could be thrown in an object's destructor method when deleted by this operator.

20.12 Remove Operator (remove)

Synopsis

The remove operator removes a value from a data structure, or, in the case the operand of the remove operator is a simple value, the value itself is removed from the variable and returned. The remove operator returns the value removed from the lvalue.

When called on a hash key, the key is removed from the hash entirely, and the value returned is the value of the key removed from the hash; when called on a list element, the element is assigned NOTHING (i.e. the list size does not change).

The remove operator will remove and return a slice from a hash when called on a hash dereferenced by a list of strings, giving the keys to remove (see example below).

The remove operator does not call destructors when operating on objects, but if removing an object from an lvalue or from a data structure within the lvalue causes the object to go out of scope, it will be destroyed, and then its destructor could throw an exception.

For a similar operator that deletes the value that is removed from the data structure, see the delete operator.

Syntax

remove *lvalue_expression*

Return Type

any

Example

```
# remove a single value from a hash
my any $var = remove $hash.value;
# remove a slice from a hash
my hash $nh = remove $h.("a", "b", "c");
```

This operator does not throw any exceptions, however exception could be thrown in an object's destructor if it goes out of scope due to the action of this operator.

20.13 Cast Operator (cast<>())

Synopsis

The cast<>() operator provides a way to tell the parser that the type of object is not actually the declared type but rather a subclass as given between the angle brackets.

Syntax

cast<*class_identifier*>(*object_expression*)

Return Type

the specific class given

Example

```
cast<SubClass>($obj).method();
```

Arguments Processed by cast<>()

Argument	Processing
class_identifier	This must be a literal unquoted string giving a class name or a namespace-qualified path to a class (ex: `Namespace::MyClass`)
object_expression	This must be an expression that evaluates to an object that can be cast to the given class; this is mostly useful at parse time to avoid non-existent-method-call warnings

Exceptions

RUNTIME-CAST-ERROR	The expression given does not evaluate to an object that can be cast to the given class

20.14 Logical Not Operator (!)

Synopsis

Reverses the logical sense of an expression (True becomes False and False becomes True).

Syntax

! *expression*

Return Type

bool

Example

```
if (!exists $error_code)
    do_something();
```

Arguments Processed by !

Argument	Processing
expression	The expression is evaluated and converted to a bool, if necessary. Then the value is logically reversed (True becomes False, False becomes True)

This operator does not throw any exceptions.

20.15 Binary Not Operator (~)

Synopsis

The value of each bit in an integer is reversed (0 becomes 1, 1 becomes 0).

Syntax

~*expression*

Return Type

int

Example

```
$a = ~$b;
```

Arguments Processed by ~

Argument	Processing
expression	The argument is converted to an integer (if necessary), and bitwise negation is performed on the argument (ex: `666 & ~27` results in `640`)

This operator does not throw any exceptions

20.16 Unary Minus Operator (-)

Synopsis

Changes the sign of numeric values.

Syntax

-expression

Return Type

int or float

Example

```
$a = -$b;
```

Arguments Processed by - (unary minus)

Argument	Processing
Float	Gives the negative of its argument as a Float (ex: `-(-1.1) = 1.1, -(1.1) = -1.1`
Integer	Gives the negative of its argument as an Integer (ex: `-(-1) = 1, -(1) = -1`

This operator does not throw any exceptions

20.17 Shift Operator (shift)

Synopsis

Removes the first element from a list and returns that element.

Syntax

shift *lvalue*

Return Type

any

Example

```
my *string $a = shift $ARGV;
```

Arguments Processed by shift

Argument	Processing
lvalue	Returns the first element of the list, and the list is modified by having the first element removed from the list. If the *lvalue* is not a list, no action is performed and the operator returns no value (NOTHING)

This operator does not throw any exceptions.

20.18 Pop Operator (pop)

Synopsis

Removes the last element from a list and returns that element.

Syntax

pop *lvalue*

Return Type

any

Example

```
$a = pop $list;
```

Arguments Processed by pop

Argument	Processing
lvalue	Returns the last element of the list, and the list is modified, having the last element removed from the list. If the *lvalue* is not a list, no action is performed and the operator returns no value (NOTHING)

This operator does not throw any exceptions.

20.19 Chomp Operator (chomp)

Synopsis

Removes the end-of-line marker(s) (`'\n'` or `'\r\n'`) from a string, or each string element in a list, or each hash key value in a hash (if the value is a string) and returns the number of characters removed.

To perform this operation on a non-lvalue expression, see the chomp() function.

Syntax

chomp*lvalue*

Return Type

int

Example

```
chomp $str;
```

Arguments Processed by chomp

Argument	Processing
lvalue (String)	Removes any EOL characters from a string and returns the number of characters removed.
lvalue (List)	Removes any EOL characters from each string element of the list passed and returns the number of characters removed.
lvalue (Hash)	Removes any EOL characters from each hash key's value (where the value is a string) and returns the number of characters removed.

This operator does not throw any exceptions.

20.20 Trim Operator (trim)

Synopsis

Removes whitespace characters from the beginning and end of a string, or each string element in a list, or each hash key value in a hash (if the value is a string) and returns the value processed (string, list, or hash).

To perform this operation on a non-lvalue expression, see the trim() function.

The following whitespace characters are removed from the beginning and end of strings: `' '` (blank spaces), `'\n'`, `'\r'`, `'\t'`, `'\v'` (vertical tab, ASCII 11), and `'\0'` (null character).

Syntax

`trim` *lvalue*

Return Type

string, list, or hash

Example

```
trim $str;
```

Arguments Processed by trim

Argument	Processing
lvalue (String)	Removes whitespace characters from the beginning and end of a string and returns the value processed.
lvalue (List)	Removes whitespace characters from the beginning and end of each string element of the list passed and returns the list.
lvalue (Hash)	Removes whitespace characters from the beginning and end of each string value of the hash passed and returns the hash.

This operator does not throw any exceptions.

20.21 Map Operator (map)

Synopsis

Executes (or maps) an expression on a list and returns the result. An optional select expression can be given to filter elements out from the result list.

If the second argument (*list_expression*) is an object inheriting the AbstractIterator class, the **map** operator iterates the object by calling AbstractIterator::next(), and the implicit argument `$1` in the *map_expression* and any optional *select_expression* is the value returned by AbstractIterator::getValue().

If the second argument is not a list or an object inheriting AbstractIterator, then *map_expression* is executed on the single value and the result is returned, and any *select_expression* is ignored.

Return Type

any

Syntax

map *map_expression*, *list_expression [, select_expression]*

Examples

```
# returns (2, 4, 6)
my list $l = map $1 * 2, (1, 2, 3);

# prints out the keys of a hash; one to a line
map printf("%s\n", $1), $hash.keyIterator();
```

Arguments Processed by map

Argument	Processing
map_expression	The expression to map on the list; the implicit argument `$1` represents the current element being processed (or the object itself if the **map** operator is iterating an object inheriting the AbstractIterator class).
list_expression	The data to process; if this is not a list then the *map_expression* is run on the single argument passed, unless the *map_expression* is an object inheriting the AbstractIterator class; in this case the **map** operator iterates the object by calling AbstractIterator::next()
[select_expression]	An optional expression than can be used to filter out elements of the list before the map expression is applied; if this expression evaluates to False on an element, then the element will be skipped and the *map_expression* will not be applied on that element.

This operator does not throw any exceptions (however note that exceptions could be thrown by expressions executed by this operator).

Since

Qore 0.8.6.2 the map operator when used with an AbstractIterator object instantiates the value returned by AbstractIterator::getValue() instead of the iterator itself

20.22 Fold Left Operator (foldl)

Synopsis

Folds an operation on a list from left to right and returns the result. The result of each individual operation is used as the first argument in the foldl expression for the next element in the list. The first operation of the fold is made by executing the fold expression on the first and second elements of the list, from this point onwards, the result of each successive operation is used as the first argument for the next operation, the second argument being the next element in the list.

If the second argument (*list_expression*) is an object inheriting the AbstractIterator class, the **`foldl`** operator iterates the object by calling AbstractIterator::next(), and the implicit arguments `$1` and `$2` in *expression* are the container values returned by AbstractBidirectionalIterator::getValue(). If the *list_expression* does not evaluate to a list or an object inheriting AbstractIterator, then the evaluated argument is returned immediately with no processing by the fold expression.

Syntax

`foldl` *expression*, *list_expression*

Return Type

any

Examples

The following returns **`foldl`** expression returns `5`

```
my int $result = foldl $1 - $2, (10, 4, 1);
```

The following **`foldl`** expression joins a list with **`", "`** or returns a single argument (if the second operand is not a list)

```
my string $str = foldl $1 + ", " + $2, $list;
```

Arguments Processed by foldl

Argument	Processing
expression	The expression to fold on the list; the implicit argument `$1` represents the result of the last operation (or the first element in the list when beginning the fold), and `$2` represents the next element of the list.
list_expression	The list, AbstractIterator object or other value according to the rules above to process

This operator does not throw any exceptions (however note that exceptions could be thrown by expressions executed by this operator).

20.23 Fold Right Operator (foldr)

Synopsis

Folds an operation on a list from right to left and returns the result. The result of each individual operation is used as the first argument in the foldr expression for the next element in the list in reverse order. The first operation of the right fold is made by executing the fold expression on the last and penultimate elements of the list, from this point onwards, the result of each successive operation is used as the first argument for the next operation, the second argument being the next element in the list in reverse order.

If the second argument (*list_expression*) is an object inheriting the AbstractBidirectionalIterator class, the **foldr** operator iterates the object by calling AbstractBidirectionalIterator::prev(), and the implicit arguments `$1` and `$2` in *expression* are the container values returned by AbstractBidirectionalIterator::getValue(). If the *list_expression* does not evaluate to a list or an object inheriting AbstractBidirectionalIterator, then the evaluated argument is returned immediately with no processing by the fold expression.

Syntax

foldr *expression*, *list_expression*

Return Type

any

Example

```
# returns -13
foldr $1 - $2, (10, 4, 1);
```

Arguments Processed by foldr

Argument	Processing
expression	The expression to fold on the list; the implicit argument `$1` represents the result of the last operation (or the last element in the list when beginning the fold), and `$2` represents the next element of the list in reverse order.
list_expression	The list, AbstractIterator object or other value according to the rules above to process

This operator does not throw any exceptions (however note that exceptions could be thrown by expressions executed by this operator).

20.24 Select From List Operator (select)

Synopsis

Selects elements from a list that meet the given criteria and returns the new list.

If the first argument (*list_expression*) is an object inheriting the AbstractIterator class, the **select** operator iterates the object by calling AbstractIterator::next(), and the implicit argument `$1` in *expression* is the value returned by AbstractIterator::getValue(); in this case the operator always returns a list (which may be empty if *expression* returns False for all elements in the iterator object or if the iterator object is empty), and the value of the object in the list returned is the value returned by AbstractIterator::getValue().

If the list expression does not evaluate to a list or an object inheriting AbstractIterator, then the select expression is evaluated using the value of the list expression as an argument, if it evalutes to true, then the value is returned, otherwise, no value is returned.

Syntax

select *list_expression*, *expression*

Return Type

any

Example

```
# returns (2, 4, 6)
select (1, 2, 3, 4, 5, 6), !($1 % 2);
```

Arguments Processed by select

Argument	Processing
list_expression	The list to process or an object inheriting AbstractIterator
expression	The expression will be evaluated on each element of the list, the implicit argument `$1` represents current element of the list (or the iterator object itself when *list_expression* is an object inheriting AbstractIterator); only if the expression evaluates to True will the element appear in the result list

This operator does not throw any exceptions (however note that exceptions could be thrown by the expression executed by this operator).

Since

Qore 0.8.6.2 the map operator when used with an AbstractIterator object instantiates the value returned by AbstractIterator::getValue() instead of the iterator itself

20.25 Elements Operator (elements)

Synopsis

Returns the number of elements in a list, the number of keys in a hash, the number of characters (not bytes) in a string, or the number of bytes in a binary object.

Syntax

`elements` *expression*

Return Type

int

Example

```
my int $size = elements $list;
```

Arguments Processed by elements

Argument	Processing
expression list	Returns the number of elements in the list
expression hash	Returns the number of keys in the hash
expression string	Returns the number of characters in the string (which may be different than the number of bytes for multi-byte character encodings such as `UTF-8`
expression binary	Returns the number of bytes in the binary object

This operator does not throw any exceptions.

See also

Qore::zzz8valuezzz9::size()

20.26 Keys Operator (keys)

Synopsis

Returns a list representing the keys in a hash.

Syntax

keys *hash_expression*

Return Type

list or NOTHING

Example

```
foreach my string $key in (keys $hash)
    printf("%s = %s\n", $key, $hash.$key);
```

Arguments Processed by keys

Argument	Processing
hash_expression	Returns a list of strings giving the keys in *hash_expression*, which must evaluate to a hash. If not, then no value is returned.

This operator does not throw any exceptions.

See also

- Qore::zzz8hashzzz9::keys()
- Qore::zzz8objectzzz9::keys()

20.27 Multiply Operator (*)

Synopsis

Multiplies two arguments.

Syntax

expression1 * *expression2*

Return Type

int, float, or number

Example

```
$value = $x * $y;
```

Arguments Processed by * (in order of precedence)

Argument	Processing
number	Gives the result of multiplying its arguments; ints and floats are converted to numbers if at least one of the arguments is a number
float	Gives the result of multiplying its arguments; ints are converted to floats if at least one of the arguments is a float
int	Gives the result of multiplying its arguments
any other type	Converts argument to a number and performs the multiplication

This operator does not throw any exceptions.

20.28 Divide Operator (/)

Synopsis

Divides a number by another.

Syntax

expression1 / *expression2*

Return Type

int, float, or number

Example

```
$value = $x / $y;
```

Arguments Processed by / (in order of precedence)

Argument	Processing
number	Gives the result of dividing its arguments; ints and floats are converted to numbers if at least one of the arguments is a number
float	Gives the result of dividing its arguments; ints are converted to floats if at least one of the arguments is a float
int	Gives the result of dividing its arguments
any other type	Converts argument to a number and performs the division

Exceptions

DIVISION-BY-ZERO	division by zero error

20.29 Modula Operator (%)

Synopsis

Gives the integer remainder after division of one number by another.

Syntax

expression1 `%` *expression2*

Return Type

int

Example

```
$mod = $x % $y;
```

Arguments Processed by %

Argument	Processing
int	Gives *expression1* modula *expression2* (ex: `12 % 10` result in `2`). Arguments are converted to integers if necessary.

This operator does not throw any exceptions.

20.30 Plus (Addition and Concatentation) Operator (+)

Synopsis

Numeric addition, list, string, binary, and hash concatenation operator.

Syntax

expression1 + *expression2*

Return Type

int, float, number, date, list, string, binary, or hash

Example

```
$a = 1 + 2;

$string = "hello" + "-there";

$list = (1, 2) + ("three", "four", "five");

$hash = ( "key1" : 1, "key2" : 2) + ( "key3" : "three", "key4": "four");

$bin = $bin1 + $bin2;
```

Arguments Processed by + (in order of precedence)

Argument	Processing
list list	Gives the result of concatenating its arguments, i.e. `(1, 2) + (3, 4) = (1, 2, 3, 4)`
string	Gives the result of concatenating its arguments
date	Gives the result of adding date/time values (see Date/Time Arithmetic)
number	Gives the result of adding its arguments
float	Gives the result of adding its arguments
int	Gives the result of adding its arguments
hash	Gives the result of concatenating/merging its arguments. Any common keys will be overwritten by the values in the second hash (*expression2*)

This operator does not throw any exceptions.

20.31 Minus Operator (-)

Synopsis

With float, integer, or number arguments, subtracts one number from another.

With date arguments, subtracts one date from another; if both date arguments are absolute dates, the result is a relative date (duration) giving the time between them; if the first date argument is an absolute date and the second is a relative date (duration), then the result is an absolute date. If both date arguments are relative dates, then the result is a relative date. If the first argument is a relative date and the second date is an absolute date, the result is an absolute date as if the operands were reversed.

However, if the left-hand side is a hash, and the right-hand side is a string, then the hash key represented by the string will be removed from the hash. If the left-hand side is a hash and the right-hand side is a list, then each element in the list will be converted to a string and any hash key with that name will be deleted from the hash.

Syntax

expression1 − *expression2*

Return Type

int, float, number, date, or hash

Example

```
$num = $x - $y;

$date = 2010-05-13 - P3MT14H10M;

$hash = $hash - "key";

$hash = $hash - ("key1", "key2", "key3");
```

Arguments Processed by - (in order of precedence)

Argument	Processing
date	date subtraction: *expression1* - *expression2*
number	arithmetic subtraction: *expression1* - *expression2*
float	arithmetic subtraction: *expression1* - *expression2*
int	arithmetic subtraction: *expression1* - *expression2*
hash - string	hash key deletion: *expression1* - *expression2*
hash - list	hash key deletion: *expression1* - *expression2*; all elements of the list are converted to strings (if necessary) and any keys with those names are deleted from the hash.

This operator does not throw any exceptions.

20.32 Shift Right Operator (>>)

Synopsis

Shifts bits in an integer towards zero (divides an integer by a power of 2)

Syntax

expression1 >> *expression2*

Return Type

int

Example

```
$a = $x >> $y;
```

Arguments Processed by >>

Argument	Processing
int	Gives the result of shifting *expression1* right by *expression2* bits. Arguments are converted to integers if necesssary.

This operator does not throw any exceptions.

20.33 Shift Left Operator (<<)

Synopsis

Shifts bits in an integer towards infinity (multiplies an integer by a power of 2)

Syntax

expression1 << *expression2*

Return Type

int

Example

```
$a = $x << $y;
```

Arguments Processed by <<

Argument	Processing
int	Gives the result of shifting *expression1* left by *expression2* bits. Arguments are converted to integers if necessary.

This operator does not throw any exceptions.

20.34 Class Instance Operator (instanceof)

Synopsis

Tests if an expression is an instance of a given class or not.

Syntax

expression **instanceof** *class_identifier*

Return Type

bool

Example

```
if ($obj instanceof Qore::Mutex)
    print("object is Mutex\n");
```

Arguments Processed by instanceof

Argument	Processing
expression	If expression is an instance of the named class, then the operator returns True, otherwise returns False. The operator will return True if the class is a base class, also even if it is privately inherited.

This operator does not throw any exceptions.

20.35 Exists Operator (exists)

Synopsis

Tests if an expression represents a value or not.

Syntax

exists *expression*

Return Type

bool

Example

```
if (exists $a)
    printf("a = $n\n", $a);
```

Arguments Processed by exists

Argument	Processing
expression	If *expression* evaluates to a value, then the operator returns True, otherwise returns False.

This operator does not throw any exceptions.

20.36 Less Than Operator (<)

Synopsis

Tests if a value is less than another; types are converted if necessary (ex: `("1" < 2)` is True).

Syntax

expression1 < *expression2*

Return Type

bool

Example

```
if ($x < $y)
    printf("%n is less than %n\n", $x, $y);
```

Arguments Processed by < (in order of precedence)

Argument	Processing
number	If *expression1* is numerically less than *expression2*, returns True, otherwise returns False
float	If *expression1* is numerically less than *expression2*, returns True, otherwise returns False
int	If *expression1* is numerically less than *expression2*, returns True, otherwise returns False
string	If *expression1* comes before *expression2* in string sort order, returns True, otherwise returns False
date	If *expression1* is before (or a shorter amount of time than of the arguments are Relative Date/Time Values (Durations)) *expression2*, returns True, otherwise returns False

This operator does not throw any exceptions.

20.37 Greater Than Operator (>)

Synopsis

Tests if a value is greater than another; types are converted if necessary (ex: `("2" > 1)` is True).

Syntax

expression1 > *expression2*

Return Type

bool

Example

```
if ($x > $y)
    printf("%n is less than %n\n", $x, $y);
```

Arguments Processed by > (in order of precedence)

Argument	Processing
number	If *expression1* is numerically greater than *expression2*, returns True, otherwise returns False
float	If *expression1* is numerically greater than *expression2*, returns True, otherwise returns False
int	If *expression1* is numerically greater than *expression2*, returns True, otherwise returns False
string	If *expression1* comes after *expression2* in string sort order, returns True, otherwise returns False
date	If *expression1* is after *expression2*, returns True, otherwise returns False

This operator does not throw any exceptions.

20.38 Equals Operator (==)

Synopsis

Tests if a value is equal to another; types are converted if necessary (ex: `("1" == 1)` is True). For absolute equals, where types must also be equal to return true, see the Absolute Equals Operator (===).

Syntax

expression1 == *expression2*

Return Type

bool

Example

```
if ($x == $y)
    printf("%n is equal to %n\n", $x, $y);
```

Arguments Processed by == (in order of precedence)

Argument	Processing
string	If *expression1* is equal to *expression2*, returns True, otherwise False
number	If *expression1* is equal to *expression2*, returns True, otherwise False
float	If *expression1* is equal to *expression2*, returns True, otherwise False
int	If *expression1* is equal to *expression2*, returns True, otherwise False

date	If *expression1* is equal to *expression2*, returns True, otherwise False
list	If each element in the each list where order is relevant satisfies this operator, the operator returns True, otherwise it returns False
hash	If each hash has the same keys and the value of each equal key in each hash satisfies this operator, the operator returns True, otherwise it returns False
binary	If *expression1's* memory contents and size are equal to *expression2's*, then returns True, otherwise False
object	If *expression1* is a reference to the same object as *expression2*, then returns True, otherwise False
NULL	If both expressions are NULL, returns True, otherwise returns False
NOTHING	If neither expression has a value, returns True, otherwise returns False

This operator does not throw any exceptions.

20.39 Not Equals Operator (!=)

Synopsis

Tests if a value is not equal to another; types are converted if necessary (ex: `("1" != 1)` is False).

Syntax

expression1 != *expression2*

Return Type

bool

Example

```
if ($x != $y)
    printf("%n is not equal to %n\n", $x, $y);
```

Arguments Processed by != (in order of precedence)

Argument	Processing
string	If *expression1* is not equal to *expression2*, returns True, otherwise False
number	If *expression1* is not equal to *expression2*, returns True, otherwise False
float	If *expression1* is not equal to *expression2*, returns True, otherwise False
int	If *expression1* is not equal to *expression2*, returns True, otherwise False
date	If *expression1* is not equal to *expression2*, returns True, otherwise False
list	If each element in the each list where order is relevant satisfies this operator, the operator returns True, otherwise it returns False

hash	If the hashes have different keys or the value of each equal key in each hash satisfies this operator, the operator returns True, otherwise it returns False
binary	If *expression1's* memory contents or size are not equal to *expression2's*, then returns True, otherwise False
object	If *expression1* is not a reference to the same object as *expression2*, then returns True, otherwise False
NULL	If either expressions is not NULL, returns True, otherwise returns False
NOTHING	If one of the expressions has a value, returns True, otherwise returns False

This operator does not throw any exceptions.

20.40 Less Than Or Equals Operator (<=)

Synopsis

Tests if a value is less than or equals to another value; types are converted if necessary (ex: `("1" <= 2)` is True).

Syntax

expression1 <= *expression2*

Return Type

bool

Example

```
if ($x <= $y)
    printf("%n is less than or equal to %n\n", $x, $y);
```

Arguments Processed by <= (in order of precedence)

Argument	Processing
number	If *expression1* is numerically less than or equal to *expression2*, returns True, otherwise returns False
float	If *expression1* is numerically less than or equal to *expression2*, returns True, otherwise returns False
int	If *expression1* is numerically less than or equal to *expression2*, returns True, otherwise returns False
string	If *expression1* comes before in string sort order or is the same as *expression2*, returns True, otherwise returns False
date	If *expression1* is before or is the same exact date and time as *expression2*, returns True, otherwise returns False

This operator does not throw any exceptions.

20.41 Greater Than Or Equals Operator (>=)

Synopsis

Tests if a value is greater than or equals to another value; types are converted if necessary (ex: `("2" >= 1)` is True).

Syntax

expression1 >= *expression2*

Return Type

bool

Example

```
if ($x >= $y)
    printf("%n is greater than or equal to %n\n", $x, $y);
```

Arguments Processed by >= (in order of precedence)

Argument	Processing
number	If *expression1* is numerically greater than or equal to *expression2*, returns True, otherwise returns False
float	If *expression1* is numerically greater than or equal to *expression2*, returns True, otherwise returns False
int	If *expression1* is numerically greater than or equal to *expression2*, returns True, otherwise returns False
string	If *expression1* comes after in string sort order or is the same as *expression2*, returns True, otherwise returns False
date	If *expression1* is after or is the same exact date and time as *expression2*, returns True, otherwise returns False

This operator does not throw any exceptions.

20.42 Comparison (<=>) Operator

Synopsis

Tests if the left-hand value is less than, equal, or greater than the right-hand value; types are converted if necessary (ex: ("1" <=> 2) returns -1).

Syntax

expression1 <=> *expression2*

Return Type

int

Example

```
switch ($x <=> $y) {
    case -1:
        print("$x is less than $y\n");
        break;

    case 0:
        print("$x is equal to $y\n");
        break;

    case 1:
        print("$x is greater than $y\n");
        break;
}
```

Arguments Processed by <=> (in order of precedence)

Argument	Processing
string	If *expression1* comes after in string sort order as *expression2*, returns `1`, otherwise if they are equal, returns `0`, otherwise if *expression1* comes before *expression2*, returns `-1`
number	If *expression1* is numerically greater than *expression2*, returns `1`, otherwise if they are equal returns `0`, otherwise returns `-1`
float	If *expression1* is numerically greater than *expression2*, returns `1`, otherwise if they are equal returns `0`, otherwise returns `-1`
int	If *expression1* is numerically greater than *expression2*, returns `1`, otherwise if they are equal returns `0`, otherwise returns `-1`
date	If *expression1* is after *expression2*, returns `1`, otherwise if they are equal returns `0`, otherwise returns `-1`

This operator does not throw any exceptions.

20.43 Absolute Equals Operator (===)

Synopsis

Checks two values for equality without doing any data type conversions; if the types do not match, then the result is False.

Syntax

expression1 === *expression2*

Return Type

bool

Example

```
if ($x === $y)
    printf("%n is equal to %n and has the same data type as well\n", $x, $y);
```

Arguments Processed by ===

Argument	Processing
All	This operator returns True only if the types and values of both sides of the operator are exactly equal, otherwise returns False. No type conversions are done.

This operator does not throw any exceptions.

20.44 Absolute Not Equals Operator (!==)

Synopsis

Checks two values for inequality without doing any data type conversions. If the data types do not match, then returns True.

Syntax

expression1 !== *expression2*

Return Type

bool

Example

```
if ($x !== $y)
    printf("%n is not equal to %n and may not have the data type as well\n", $x, $y);
```

Arguments Processed by !==

Argument	Processing
All	This operator returns True if either the types or the values of the arguments are different, otherwise it retuns False. No type conversions are done.

This operator does not throw any exceptions.

20.45 Regular Expression Match Operator (=~)

Synopsis

Checks for a regular expression match; returns True if the expression matches the string, False if not. See regex_options for the meaning of the `i`, `s`, `x`, and `m` options after the regular expression.

See Regular Expressions for more information about regular expression support in Qore.

Syntax

expression =~ `[m]/`*regex*`/[isxm]`

Return Type

bool

Example

```
if ($str =~ /hello/)
    printf("%s contains 'hello'\n", $str);
```

Arguments Processed by =~

Argument	Processing
string	This operator returns True if the regular expression in *regex* matches the string in *expression*.

This operator does not throw any exceptions.

20.46 Regular Expression No Match Operator (!~)

Synopsis

Checks for a regular expression non match; returns True if the expression does not match the string, False if it does. See regex_options for the meaning of the `i`, `s`, `x`, and `m` options after the regular expression.

See Regular Expressions for more information about regular expression support in Qore.

Syntax

expression !~ `[m]/`*regex*`/[isxm]`

Return Type

bool

Example

```
if ($str !~ /hello/)
    printf("%s does not contain 'hello'\n", $str);
```

Arguments Processed by !~

Argument	Processing
string	This operator returns True if the regular expression in *regex* does not match the string in *expression*.

This operator does not throw any exceptions.

20.47 Regular Expression Substitution Operator

Synopsis

Looks for a regular expression match in a string, and, if found, substitutes the matched string with a new string. Subpattern backreferences are supported in the target string, `$1`=first subpattern, `$2`=second subpattern, etc... See regex_options for the meaning of the `i`, `s`, `x`, and `m` options after the regular expression.

See Regular Expressions for more information about regular expression support in Qore.

Syntax

lvalue =~ `s/` *regex_pattern* `/` *target_string* `/[isxmg]`

Return Type

string or NOTHING (if the lvalue does not hold a string)

Example

```
$str =~ s/hello/goodbye/i;
$str =~ s/(\w+) +(\w+)/$2, $1/;
```

Arguments Processed by =~ s///

Argument	Processing
string	This operator substitutes text in the *lvalue* string if the regular expression matches. Subpattern backreferences are supported in *target_string*, `$1`=first subpattern, `$2`=second subpattern, etc..

This operator does not throw any exceptions.

20.48 Regular Expression Pattern Extraction Operator

Synopsis

Matches regular expression patterns (enclosed in parentheses) in a string and returns a list giving the text matched for each pattern. If the regular expression does not match, then no value (NOTHING) is returned. See regex_options for the meaning of the `i`, `s`, `x`, and `m` options after the regular expression.

See Regular Expressions for more information about regular expression support in Qore.

Syntax

string =~ x/*regex_with_patterns*/[isxm]

Return Type

list or NOTHING (if the lvalue does not hold a string or if the pattern is not matched)

Example

```
$list =~ x/(\w+):(\w+)/;
$list =~ x/(.*)\.(.*)/;
```

Arguments Processed by =~ x//

Argument	Processing
string	This operator extracts strings from *string* based on patterns enclosed in parentheses in the regular expression.

This operator does not throw any exceptions.

20.49 Transliteration Operator

Synopsis

Makes character substitutions in an lvalue; character ranges can also be used.

Syntax

lvalue =~ tr/*source_chars*/*target_chars*/

Return Type

string or NOTHING (if the lvalue does not hold a string)

Example

```
$str =~ tr/a-z/A-Z/;
```

Arguments Processed by =~ tr//

Argument	Processing
string	This operator substitutes characters in the *lvalue* string. Note that if there are more characters in *source_chars* than in *target_chars*, then the last character in *target_chars* will be used for any source matches where the source character position is greater than the length of *target_chars*.

This operator does not throw any exceptions.

20.50 Bitwise/Binary And Operator (&)

Synopsis

Performs a bitwise (binary) AND operation on two integers.

Syntax

expression1 & *expression2*

Return Type

int

Example

```
$a = $x & $y;
```

Arguments Processed by &

Argument	Processing
int	Gives the result of the binary (bitwise) AND operation between *expression1* and *expression2* (ex: `0xffb2 & 0xa1 = 0xa1`); operands are converted to integers if necessary.

This operator does not throw any exceptions.

20.51 Bitwise/Binary Or Operator (|)

Synopsis

Performs a bitwise (binary) OR operation on two integers.

Syntax

expression1 | *expression2*

Return Type

int

Example

```
$a = $x | $y;
```

Arguments Processed by |

Argument	Processing	
int	Gives the result of the binary (bitwise) OR operation between *expression1* and *expression2* (ex: `0xb001`	`0xfea = 0xbfeb`); operands are converted to integers if necessary

This operator does not throw any exceptions.

20.52 Bitwise/Binary Xor Operator (^)

Synopsis

Performs a bitwise (binary) XOR operation on two integers.

Syntax

expression1 ^ *expression2*

Return Type

int

Example

```
$a = $x ^ $y;
```

Arguments Processed by ^

Argument	Processing
int	Gives the result of the binary (bitwise) EXCLUSIVE OR operation between *expression1* and *expression2* (ex: `0xaef1 & 0xfb32 = 0x55c3`); operands are converted to integers if necessary

This operator does not throw any exceptions.

20.53 Logical And Operator (&&)

Synopsis

Checks to see if two expressions are True with logical short-circuiting.

Syntax

expression1 && *expression2*

Return Type

bool

Example

```
if ($x && $y)
    printf("%n and %n are both True\n", $x, $y);
```

Arguments Processed by &&

Argument	Processing
bool	Returns True if both expressions are True, False if otherwise. Logical short-circuiting is implemented; if *expression1* is False, then *expression2* is not evaluated, and the operator returns False.

This operator does not throw any exceptions.

20.54 Logical Or Operator (||)

Synopsis

Returns True if either of the arguments are True with logical short-circuiting.

Syntax

expression1 || *expression2*

Return Type

bool

Example

```
if ($x || $y)
    printf("either %n or %n or both are True\n", $x, $y);
```

Arguments Processed by ||

Argument	Processing
bool	Returns True if either or both expressions evaluate to True, False if otherwise. Logical short-circuiting is implemented; if *expression1* is True, then *expression2* is not evaluated, and the operator returns True.

This operator does not throw any exceptions.

20.55 Conditional Operator (? :)

Synopsis

Evaluates and returns the value of one of two expressions depending on the value of a conditional expression.

Syntax

expression ? *if_true_expression* : *if_false_expression*

Return Type

any

Example

```
$a = ($z > 100 ? "Big" : "Small");
```

Arguments Processed by ? :

Argument	Processing
All	If *expression* is evaluated to be True, then the *if_true_expression* is evaluated and returned. Otherwise the *if_false_expression* is evaluated and returned.

This operator does not throw any exceptions.

20.56 Comma Operator (,)

Synopsis

Makes a list from more than one element.

Syntax

expression1, *expression2*

Return Type

list

Example

```
$a = 1, 2, "three";
```

Arguments Processed by ,

Argument	Processing
All	The comma operator builds lists of arguments

This operator does not throw any exceptions.

20.57 Unshift Operator (unshift)

Synopsis

Inserts an element into the first position of a list and moves all other elements up one position.

Syntax

unshift *lvalue*, *expression*

Return Type

any

Example

```
unshift $list, "one";
```

Arguments Processed by unshift

Argument	Processing
All	Inserts the value of *expression* as the first element in the list given by *lvalue*. All other elements in the list are moved forward.

20.58 Push Operator (push)

Synopsis

Adds one element to the end of a list and returns the list processed (or NOTHING if the *lvalue* is not a list).

Syntax

push *lvalue*, *expression*

Return Type

list or NOTHING (if the lvalue is not a list)

Example

```
push $list, "last";
```

Arguments Processed by push

Argument	Processing
All	Appends the value of the *expression* as the last element in the list given by *lvalue*. If *expression* evaluates to a list, this list will be appended as the last element of *lvalue*. To concatenate lists, use the plus operator.

20.59 Splice Operator (splice)

Synopsis

Removes and optionally inserts elements in lists, strings, and binary objects and returns the lvalue after processing. For a similar operator that returns the values removed, see the extract operator.

Works on strings, lists, and binary data in a similar way; removes elements from a list, characters from a string, or bytes from binary data and optionally inserts new ones. If no *length_expression* is given, splice removes all elements/characters/bytes from the list, string, or binary data lvalue starting at *offset_expression* (offsets begin at 0). Otherwise, a number of elements/characters/bytes equal to *length_expression* is removed (or up to the end of the list/string/data if applicable). If *substitution_expression* is present, then the removed elements/characters/bytes are substituted with the elements/string/bytes given by this expression.

Note that string splice takes character offsets, which may not be the same as byte offsets for multi-byte character encodings, such as `UTF-8`

Syntax

`splice` *lvalue*, *offset_expression, [length_expression, [substitution_expression]]*

Return Type

list, string, or binary (returns *lvalue* after processing)

Example

```
splice $list, 2, 2;

splice $string, 2, 2, "-text-";

splice $bin, 2, 2, <deadbeef>;
```

Arguments Processed by splice

Argument	Processing
lvalue (list, string, or binary)	If the lvalue is a list, list elements are processed, if it is a string, characters in the string are processed, and for binary data, bytes are processed. For any other data type, no action is taken.
offset_expression	The start element/character/byte position for removing elements/characters/bytes from the list, string, or binary data; if this value is negative, it gives the element offset from the end of the data
length_expression	The number of elements/characters/bytes to remove. If this expression is not present, then all elements/characters/bytes from the offset to the end of the list/string/binary data are removed. If this expression is present and evaluates to 0, no characters/elements/byte are removed; if this value is negative, then it gives an offset from the end of the data (ie -2 means remove all elements/characters/bytes up to but not including the last two)

substitution_expression	For list splice, an optional element or list to substitute for the removed elements (to insert a list in a single element's position, make sure that the list to be inserted is the first and only element of another list used as the argument in this position; in other words, pass a list within a single-element list). For string splice, an optional string to substitute for the removed characters. For binary splice, string or binary data to substitute for any removed bytes.

20.60 Extract Operator (extract)

Synopsis

Removes and optionally inserts elements in lists and strings. For a similar operator that removes values from an lvalue and returns the lvalue (instead of the value removed), see the splice operator.

Works on either strings, lists, and binary data in a similar way; removes elements from a list, characters from a string, and bytes from binary data and optionally inserts new ones. If no length_expression is given, extract removes all elements/characters/bytes from the lvalue starting at *offset_expression* (offsets begin at 0). Otherwise, a number of elements/characters/bytes equal to *length_expression* is removed (or up to the end of the data if applicable). If *substitution_expression* is present, then the removed elements/characters/bytes are substituted with the data given by this expression.

When operating on lists, a list is returned of any elements extracted (if no elements are extracted, then an empty list is returned); when operating on strings, a string is extracted of all characters extracted from the string (if no characters are extracted, then an empty string is returned). When operating on binary data, a binary object is returned.

Note that string extract takes character offsets, which may not be the same as byte offsets for multi-byte character encodings, such as `UTF-8`

Syntax

`extract` *lvalue*, *offset_expression, [length_expression, [substitution_expression]]*

Return Type

list, string, or binary (the value(s) removed from *lvalue*)

Example

```
my list $sublist = extract $list, 2, 2;

my string $substring = extract $string, 2, 2, "-text-";

my binary $b = extract $bin, 2, 2, <deadbeef>;
```

Arguments Processed by extract

Argument	Processing
lvalue (list, string, or binary)	If the lvalue is a list, list elements are processed, if it is a string, characters in the string are processed, and for binary data, bytes are processed. For any other data type, no action is taken.

offset_expression	The start element/character/byte position for removing elements/characters/bytes from the list, string, or binary data; if this value is negative, it gives the element offset from the end of the data
length_expression	The number of elements/characters/bytes to remove. If this expression is not present, then all elements/characters/bytes from the offset to the end of the list/string/binary data are removed. If this expression is present and evaluates to 0, no characters/elements/byte are removed; if this value is negative, then it gives an offset from the end of the data (ie -2 means remove all elements/characters/bytes up to but not including the last two)
substitution_expression	For list extract, an optional element or list to substitute for the removed elements (to insert a list in a single element's position, make sure that the list to be inserted is the first and only element of another list used as the argument in this position; in other words, pass a list within a single-element list). For string splice, an optional string to substitute for the removed characters. For binary splice, string or binary data to substitute for any removed bytes.

20.61 Assignment Operator (=)

Synopsis

Assigns a value to an lvalue and returns the value assigned.

Syntax

lvalue = *expression*

Return Type

any

Example

```
$a = 1;
```

Arguments Processed by =

Argument	Processing
All	Assigns the value of *expression* to *lvalue*

20.62 Plus Equals Operator (+=)

Synopsis

Increments and concatentates an lvalue with the value of an expression depending on the data type of the lvalue, unless the lvalue is NOTHING, in which case this operator acts like the assignment operator (simply assigns the value of the right hand side to the lvalue).

Syntax

lvalue += *expression*

Return Type

int, float, number, date, list, string, binary, hash, or object

Example

```
$a += 10;

$date += P1M2DT45M;

$list += $new_element;

$string += ".foo";

$binary += <0c67a374>

$hash += ("new-key" : 1, "other" : "two");

$object += $hash;
```

Arguments Processed by +=

Argument	Processing
lvalue (list)	the expression will be evaluated and concatenated to the lvalue. If *expression* is a list, the lists will be concatenated, to ensure adding a single element to a list, use the push operator
lvalue (hash or (object)	the *expression* will be evaluated, and, if it is a hash or object, then it's members will be added to the *lvalue*, any duplicate elements in the *lvalue* will be overridden by elements in the *expression*.
lvalue (string)	the *expression* will be evaluated and converted to a string if necessary and concatenated to the *lvalue*.
lvalue (number)	the *expression* will be evaluated and converted to a number if necessary and added to the *lvalue*.
lvalue (float)	the *expression* will be evaluated and converted to a float if necessary and added to the *lvalue*.
lvalue (binary)	the *expression* will be evaluated and converted to a binary if necessary and added to the *lvalue*.
lvalue (date)	the *expression* will be evaluated and converted to a date if necessary and added to the *lvalue*.
lvalue (NOTHING)	the *lvalue* will be assigned to the value of *expression*.
lvalue (all other types)	the *lvalue's* type will be converted to an integer, and the *expression* will be evaluated and converted to an integer if necessary, and then the result will be added to the *lvalue*.

20.63 Minus Equals Operator (-=)

Synopsis

For a float or integer argument, decrements the value of an *lvalue* by the value of an *expression*. However if the *lvalue* is a hash or object and the *expression* is a string, removes the key represented by the string from the hash or object.

Syntax

lvalue −= *expression*

Return Type

int, float, number, date, hash, or object

Example

```
$a -= 10;

$date -= PT45H213S;

$hash -= "key";

$hash -= ("key1", "key2");

$object -= "key";

$object -= $list_of_keys;
```

Arguments Processed by -=

Argument	Processing
lvalue (number)	the *expression* will be evaluated and converted to a number if necessary and subtracted from the *lvalue*
lvalue (float)	the *expression* will be evaluated and converted to a float if necessary and subtracted from the *lvalue*
lvalue (date)	the *expression* will be evaluated and converted to a date if necessary and subtracted from the *lvalue*
lvalue (hash or (object), *expression* (string)	the hash key represented by *expression* will be removed from the *lvalue*
lvalue (hash or (object), *expression* (list)	each element in the list will be converted to a string (if necessary) and the key represented by each string will be removed from the hash or object
lvalue (NOTHING), *expression* (any type)	the *expression* will be assigned to *lvalue*
lvalue (all other types)	the *lvalue's* type will be converted to an integer (if necessary), and the *expression* will be evaluated and converted to an integer (if necessary), and then the result will be subtracted from the *lvalue*

20.64 And Equals Operator (&=)

Synopsis

Performs a bitwise (binary) AND operation on an *lvalue* using the value of an *expression* and returns the new value.

Syntax

lvalue `&=` *expression*

Return Type

int

Example

```
$a &= 0xfe;
```

Arguments Processed by &=

Argument	Processing
All	the *lvalue's* type will be converted to an integer if necessary, and the *expression* will be evaluated and converted to an integer as well if necessary, and then the result will be binary and'ed to the lvalue

20.65 Or Equals Operator (|=)

Synopsis

Performs a bitwise (binary) OR operation on an *lvalue* using the value of an *expression* and returns the new value.

Syntax

lvalue |= *expression*

Return Type

int

Example

```
$a |= 0xba;
```

Arguments Processed by |=

Argument	Processing
All	the *lvalue's* type will be converted to an integer if necessary, and the *expression* will be evaluated and converted to an integer as well if necessary, and then the result will be binary or'ed to the *lvalue*

20.66 Modula Equals Operator (%=)

Synopsis

Performs a modula calculation on an lvalue using the value of an expression and returns the new value.

Syntax

lvalue %= *expression*

Return Type

int

Example

```
$a %= 100;
```

Arguments Processed by %=

Argument	Processing
All	the *lvalue's* type will be converted to an integer if necessary, and the *expression* will be evaluated and converted to an integer as well if necessary, and then the result will be used to divide the *lvalue's* value and the remainder will be saved to the *lvalue*

20.67 Multiply Equals Operator (*=)

Synopsis

Performs a multiplication operation on an lvalue using the value of an expression and returns the value assigned.

Syntax

lvalue *= *expression*

Return Type

int, float, or number

Example

```
$a *= 10;
```

Arguments Processed by *=

Argument	Processing
All	The type precedence from highest to lowest is number, float float, and int, other types are converted to int. The *expression* will be evaluated and multiplied by the *lvalue*, and the result will be saved to the *lvalue*.

20.68 Divide Equals Operator (/=)

Synopsis

Performs a division operation on an lvalue using the value of an expression and returns the value assigned.

Syntax

lvalue /= *expression*

Return Type

int, float, or number

Example

```
$a /= 10;
```

Arguments Processed by /=

Argument	Processing
All	The type precedence from highest to lowest is number, float float, and int, other types are converted to int; The *expression* will be evaluated and multiplied by the *lvalue*, and the result will be saved to the *lvalue*. The *expression* will be evaluated and used to divide the *lvalue*, and the result will be saved to the *lvalue*.

Exceptions

DIVISION-BY-ZERO	If the divisor expression evaluates to zero, this exception is thrown

20.69 Xor Equals Operator (^=)

Synopsis

Performs an exclusive-or operation on an lvalue using the value of an expression.

Syntax

lvalue ^= *expression*

Return Type

int

Example

```
$a ^= 0xf9034ba7;
```

Arguments Processed by ^=

Argument	Processing
All	Values are converted to integers if necessary. The *expression* will be evaluated and exclusive-or'ed with the *lvalue*, and the result will be saved to the *lvalue*

20.70 Shift Left Equals Operator (<<=)

Synopsis

Performs a shift-left operation on an lvalue using the value of an expression and returns the value assigned.

Syntax

lvalue <<= *expression*

Return Type

int

Example

```
$a <<= 3;
```

Arguments Processed by <<=

Argument	Processing
All	Values are converted to integers if necessary. The *expression* will be evaluated and this value will determine how many bits the *lvalue* will be shifted left. The result will be saved to the *lvalue*.

20.71 Shift Right Equals Operator (>>=)

Synopsis

Performs a shift-right operation on an lvalue using the value of an expression and returns the value assigned.

Syntax

lvalue >>= *expression*

Return Type

int

Example$

```
$a >>= 3;
```

Arguments Processed by >>=

Argument	Processing
All	Values are converted to integers if necessary. The *expression* will be evaluated and this value will determine how many bits the *lvalue* will be shifted right. The result will be saved to the *lvalue*.

Chapter 21

Regular Expressions

Regular expression functionality in Qore is provided by PCRE: Perl-Compatible Regular Expression library.

Using this library, Qore implements regular expression pattern matching using the same syntax and semantics as Perl 5.

The following is a list of operators based on regular expressions (or similar to regular expressions in the case of the transliteration operator).

Regular Expression Operators

Operator	Description
Regular Expression Match Operator (=~)	Returns True if the regular expression matches a string
Regular Expression No Match Operator (!~)	Returns True if the regular expression does not match a string
Regular Expression Substitution Operator	Substitutes text in a string based on matching a regular expression
Regular Expression Pattern Extraction Operator	Returns a list of substrings in a string based on matching patterns defined by a regular expression
Transliteration Operator	Not a regular expression operator; transliterates one or more characters to other characters in a string

See the table below for valid regular expression options.

Regular Expression Options

Option	Description
i	Ignores case when matching
m	makes start-of-line (^) or end-of-line ($) match after or before any newline in the subject string
s	makes a dot (.) match a newline character
x	ignores whitespace characters and enables comments prefixed by #
g	makes global substitutions or global extractions (only applicable with the substitution and extraction operators)

The following is a list of functions providing regular expression functionality where the pattern may be given at run-time:

Regular Expression Functions

Function	Description

regex()	Returns True if the regular expression matches a string
regex_subst()	Substitutes a pattern in a string based on regular expressions and returns the new string
regex_extract()	Returns a list of substrings in a string based on matching patterns defined by a regular expression

Chapter 22

Date/Time Arithmetic

Date/time arithmetic is relatively straightforward and should normally produce the expected results. However with leap years, months with different lengths, and daylights savings time the situation can be confusing; this section will clarify how Qore does date arithmetic considering these special cases.

22.1 Adding and Subtracting Years and Months

Adding or subtracting years and months (ex: `$date += 2Y + 3M`) will give you the same day on the desired month in the desired year. If the target month has fewer days than the source month, then you will get the last day of the month in that year. For example:

```
prompt% qore -X '2004-02-29Z - 1Y'
2003-02-28 00:00:00 Fri Z (UTC)
```

22.2 Adding and Subtracting Days

Adding or subtracting days means adding or subtracting 24h periods; i.e. you will get the same time in the result of subtracting days, for example:

```
prompt% qore -X '2004-02-29T10:15:00Z - 10D'
2004-02-19 10:15:00 Thu Z (UTC)
```

22.3 Finding the Difference Between Two Dates

Subtracting one absolute date from another will result in a relative date, normalized to the hour (that is, microseconds over 999,999 are converted to seconds, seconds over 59 to minutes, and minutes over 59 to hours; days, months, and years will not appear in the result as they do not indicate a fixed period of time but rather can vary in length depending on the absolute date/time starting point. For example:

```
prompt% qore -X '2007-02-29T10:15:03.255Z - 2004-02-29T10:14:02.100Z'
<time: 26304 hours 1 minute 1 second 155 milliseconds>
```

To find the difference in seconds between two dates, convert each date value to an integer and subtract as follows:

```
prompt% qore -X 'int(2004-02-29Z) - int(2004-02-28Z)'
86400
```

Or use the get_duration_seconds() function as follows:

```
prompt% qore -X 'get_duration_seconds(2004-02-29Z - 2004-02-28Z)'
86400
```

22.4 Timezones and Daylight Savings Time

Time zones and daylight savings time information is supplied by the system's zoneinfo database (if any exists; see Time Zone Handling for more information).

To find out if the current time zone has daylight savings time, execute the following:

```
prompt% qore -X 'TimeZone::get().hasDST()'
True
```

See the Qore::TimeZone class for more information on time zone information.

22.5 Leap Years and the Gregorian Calendar

Qore is capable of representing and performing calculations on dates before the adoption of the Gregorian calendar (proposed in 1582 and adopted at various times in Europe after this point). However all calculations are made as if the Gregorian calendar were always in effect (Qore implements a `proleptic Gregorian calendar`).

Chapter 23

Statements

Non-block statements in Qore are always terminated by a semi-colon "**;**" as in Perl, C, C++, or Java. Statements can be grouped into blocks, which are delimited by curly brackets "{" and "}" containing zero or more semi-colon delimited statements, as in C or Java. Like C, C++, and Java, but unlike perl, any Qore statement taking a statement modifier will accept a single statement or a statement block.

A statement can be any of the following (note that statements are also recursively defined):

Qore Statements

Type	Examples	Reference

An expression that changes an lvalue	`$var = 1;`
	`$var += 5;`
	`$var[1].count++;`
	`shift $var.key[$i];`
		Expressions

An expression with the new operator	`new ObjectClass(1, 2, 3);`	New Object Operator (new)
An expression with the background operator	`background function();`	Background Operator (background)
A call reference or closure call	`$call_reference($arg1, $arg2);`	Call References, Closures
A method call	`$object.method(1, 2, 3);`	Classes
An if statement	`if ($var == 3) {}`	if and else Statements
An if ... else statement	`if ($var == 3) {} else {}`	if and else Statements
A while statement	`while ($var < 10) {}`	while Statements

A do while statement	`do {} while (True);`	do while Statements
A for statement	`for (my int $i = 0; $i < 10; ++ $i) {}`	for Statements
A foreach statement	`foreach my softint $i in ($list) {}`	foreach Statements
A switch statement	`switch ($var) { case =~ /error/: throw "ERROR", $var; default: printf("%s\n", $var); }`	switch Statements
A return statement	`return $val;`	return Statements
A local variable declaration	`my string $var;` `my (int $a, string $b, bool $c);`	Variables, Variable Declarations and Lexical Scope
A global variable declaration	`our int $var;` `our (float $a, int $b, hash $c);`	Variables, Variable Declarations and Lexical Scope
A function call	`calculate($this, $that, $the_other);`	function_library
A continue statement	`continue;`	continue Statements
A break statement	`break;`	break Statements
A statement block	`{}`	zero or more statements enclosed in curly brackets
A throw statement	`throw "ERROR", $description;`	throw Statements
A try and catch statement	`try { func(); } catch (hash $ex) { printf("%s:%d: %s: %s\n", $ex.file, $ex.line, $ex.err desc); }`	try and catch Statements
A rethrow statement	`rethrow;`	rethrow Statements
A thread_exit statement	`thread_exit;`	thread_exit Statements
A context statement	`context top ($q) {}`	context Statements

A summarize statement	`summarize ($q) by (%date) where (%id != NULL) {}`	summarize Statements
A subcontext statement	`subcontext where (%type == "INBOUND" ) {}`	subcontext Statements
An on_exit statement	`on_exit $l.unlock();`	on_exit Statements
An on_success statement	`on_success $ds.commit();`	on_success Statements
An on_error statement	`on_error $ds.rollback();`	on_error Statements

23.1 if and else Statements

Synopsis

The **if** statement allows for conditional logic in a Qore program's flow; the syntax is similar to that of C, C++, or Java.

Syntax

if (*expression*)
statement
[**else**
statement]

Description

Qore if statements work like if statements in C or Java. If the result of evaluating the expression converted to a Boolean value is True, then the first statement (which can also be a block) is executed. If the result is False, and there is an **else** keyword after the first statement, the following statement is executed.

Note

Any expression that evaluates to a non-zero integer value will be converted to a Boolean True. Any expression that evaluates to zero value is interpreted as False. This is more like C and Java's behavior and not like Perl's (where any non-null string except "0" is True). To simulate Perl's boolean evaluation, use Qore::zzz8valuezzz9::val().

23.2 for Statements

Synopsis

The Qore **for** statement is most similar to the for statement in C and Java, or the non array iterator for statement in Perl. This statement is ideal for loops that should execute a given number of times, then complete. Each of the three expressions in the for statement is optional and may be omitted. To iterate through a list without directly referencing list index values, see the foreach statement.

Syntax

for (*[initial_expression]*; *[test_expression]*; *[iterator_expression]*)
statement

Description

[initial_expression]
The *initial_expression* is executed only once at the start of each for loop. It is typically used to initialize a loop variable.

[test_expression]
The *test_expression* is executed at the start of each for loop iteration. If this expression evaluates to Boolean False, the loop will terminate.

[iterator_expression]
The *iterator_expression* is executed at the end of each for loop iteration. It is typically used to increment or decrement a loop variable that will be used in the test_expression.

Example

Here is an example of a for loop using a local variable:

```
for (my int $i = 0; $i < 10; $i++)
    print("%d\n", $i);
```

23.3 foreach Statements

Synopsis

The Qore **foreach** statement is most similar to the **for** or **foreach** array iterator statement in Perl. To iterate an action until a condition is True, use the for statement instead.

Syntax

foreach [**my**|**our**] [*type*] $variable **in** (*expression*)
statement

Description

If *expression* does not evaluate to a list, then the variable will be assigned the value of the expression evaluation and the statement will only execute one time. Otherwise the variable will be assigned to each value of the list and the statement will be called once for each value.

If *expression* evaluates to an object inheriting the AbstractIterator class, the **foreach** operator iterates the object by calling AbstractIterator::next(), and the alues assigned to the iterator variable on each iteration are the container values returned by AbstractIterator::getValue(). If *expression* evalutes to NOTHING (no value); then the loop is not executed at all.

Example

Here is an example of a foreach loop using a local variable:

```
# if $str_list is a list of strings, this will remove all whitespace from the
# strings; the reference in the list expression ensures that changes
# to the iterator variable are written back to the list
foreach my string $str in (\$str_list)
    trim $str;
```

Here is an example of a foreach loop using an object derived from AbstractIterator:

```
my hash $h = ("a": 1, "b": 2);
foreach my int $i in (new HashIterator($h, True))
    printf("%s = %y\n", $i.key, $i.value);
```

Note

If a reference (`\$lvalue_expression`) is used as the list expression, any changes made to the **`foreach`** iterator variable will be written back to the list (in which case any AbstractIterator object is not iterated; references cannot be used with AbstractIterator objects as such objects provide read-only iteration).

See also

Map Operator (map) for a flexible way to iterate a list or AbstractIterator object in a single expression

23.4 switch Statements

Synopsis

The Qore switch statement is similar to the switch statement in C and C++, except that the case values can be any expression that does not need run-time evaluation and can also be expressions with simple relational operators or regular expressions using the switch value as an implied operand.

Syntax

`switch` (*expression*) {
`case` *case_expression*`:`
[statement(s)...]
[**`default`**`:`
[statement(s)...]]
}

Example

```
switch ($val) {
    case < -1:
        printf("less than -1\n");
        break;
    case "string":
        printf("string\n");
        break;
    case > 2007-01-22T15:00:00:
        printf("greater than 2007-01-22 15:00:00\n");
        break;
    case /abc/:
        printf("string with 'abc' somewhere inside\n");
        break;
    default:
        printf("default\n");
        break;
}
```

Description

The first *expression* is evaluated and then compared to the value of each *case_expression* in declaration order until one of the *case_expressions* matches or is evaluated to True. In this case all code up to a break statement is executed, at which time execution flow exits the **`switch`** statement.

Unless relational operators are used, the comparisons are "hard" comparisons; no type conversions are done, so in order for a match to be made, both the value and types of the expressions must match exactly. When relational operators are used, the operators are executed exactly as they are in the rest of Qore, so type conversions may be performed if nesessary. The only exception to this is when both arguments are strings then a soft comparison is made in order to avoid the case that strings fail to match only because their encodings are different.

To use soft comparisons, you must explicitly specify the soft equals operator as follows:

```
switch (1) {
    case == "1": print("true\n"); break;
}
```

If no match is found and a default label has been given, then any statements after the default label will be executed. If a match is made, then the statements following that case label are executed.

To break out of the switch statement, use the break statement.

As with C and C++, if no break or return statement is encountered, program control will continue to execute through other **case** blocks until on of the previous statements is encountered or until the end of the **switch** statement.

Valid Case Expression Operators

Operator	Description
>	Greater Than Operator (>)
>=	Greater Than Or Equals Operator (>=)
<	Less Than Operator (<)
<=	Less Than Or Equals Operator (<=)
==	Equals Operator (==) (with type conversions)
~=	Regular Expression Match Operator (=~) (in this case the regular expression may be optionally given without the operator)
!~	Regular Expression No Match Operator (!~)

23.5 while Statements

Synopsis

while statements in Qore are similar to while statements in Perl, C and Java. They are used to loop while a given condition is True.

Syntax

while (*expression*)
statement

Description

First the expression will be evaluated; if it evaluates to True, then statement will be executed. If it evaluates to False, the loop terminates.

Example

```
my int $a = 1;
while ($a < 10)
    $a++;
```

23.6 do while Statements

Synopsis

do while statements in Qore are similar to do while statements in C. They are used to guarantee at least one iteration and loop until a given expression evaluates to False.

Syntax

do
statement
while (*expression*);

Description

First, the *statement* will be executed, then the *expression* will be evaluated; if it evaluates to True, then the loop iterates again. If it evaluates to False, the loop terminates.

The difference between **do while** statements and while statements is that the **do while** statement evaluates its loop expression at the end of the loop, and therefore guarantees at least one iteration of the loop.

Example

```
$a = 1;
do
    $a++;
while ($a < 10);
```

23.7 continue Statements

Synopsis

Skips the rest of a loop and jumps right to the evaluation of the iteration expression.

Syntax

continue;

Description

The **continue** statement affects loop processing; that is; it has an affect on for, foreach, while, do while, context, summarize, and subcontext loop processing.

When this statement is encountered while executing a loop, execution control jumps immediately to the evaluation of the iteration expression, skipping any other statements that might otherwise be executed.

23.8 break Statements

Synopsis

Exits immediately from a loop statement or switch block.

Syntax

break;

Description

The **break** statement affects loop processing; that is; it has an affect on for, foreach, while, do while, context, summarize, and subcontext loop processing as well as on switch block processing.

When this statement is encountered while executing a loop, the loop is immediately exited, and execution control passes to the next statement outside the loop.

23.9 throw Statements

Synopsis

In order to throw an exception explicitly, the **throw** statement must be used.

Syntax

throw expression;

Description

The expression will be passed to the **catch** block of a try/catch statement, if the **throw** is executed in a try block. Otherwise the default system exception handler will be run and the currently running thread will terminate.

Qore convention dictates that a direct list is thrown with at least two string elements, the error code and a description. All system exceptions have this format.

See try/catch statements for information on how to handle exceptions, and see Exception Handling for information about how throw arguments are mapped to the exception hash.

23.10 try and catch Statements

Synopsis

Some error conditions can only be detected and handled using exception handlers. To catch exceptions, **try** and **catch** statements have to be used. When an exception occurs while executing the **try** block, execution control will immediately be passed to the **catch** block, which can capture information about the exception.

Syntax

try
statement
catch ([*$exception_hash_variable*])
statement

Description

A single variable can be specified in the catch block to be instantiated with the exception hash, giving information about the exception that has occurred. For detailed information about the exception hash, see Exception Handling.

If no variable is given in the **catch** declaration, it will not be possible to access any information about the exception in the **catch** block. However, the rethrow statement can be used to rethrow exceptions at any time in a **catch** block.

23.11 rethrow Statements

Synopsis

A **rethrow** statement can be used to rethrow an exception in a **catch** block. In this case a entry tagged as a rethrow entry will be placed on the exception call stack.

Syntax

rethrow;

Description

The rethrown exception will be either passed to the next higher-level **catch** block, or to the system default exception handler, as with a throw statement.

This statement can be used to maintain coherent call stacks even when exceptions are handled by more than one **catch** block (for detailed information about the exception hash and the format of call stacks, see Exception Handling).

Note that it is an error to use the **rethrow** statement outside of a **catch** block.

23.12 thread_exit Statements

Synopsis

thread_exit statements cause the current thread to exit immediately. Use this statement instead of the exit() function when only the current thread should exit.

Syntax

```
thread_exit;
```

Description

This statement will cause the current thread to stop executing immediately.

23.13 context Statements

Synopsis

To easily iterate through multiple rows in a hash of arrays (such as a query result set returned by the Qore::SQL::Datasource::select() or Qore::SQL::SQLStatement::fetchColumns() methods), the **context** statement can be used. Column names can be referred to directly in expressions in the scope of the context statement by preceding the name with a "%" character.

Syntax

context *[name]* (*data_expression*)
[**where** (*where_expression*)]
[**sortBy** (*sort_expression*)]
[**sortDescendingBy** (*sort_descending_expression*)]
statement

Description

data_expression
This must evaluate to a hash of arrays in order for the **context** statement to execute.

where_expression
An optional **where** expression may be given, in which case for each row in the hash, the expression will be executed, and if the where expression evaluates to True, the row will be iterated in the context loop. If this expression evaluates to False, then the row will not be iterated. This option is given so the programmer can create multiple views of a single data structure (such as a query result set) in memory rather than build different data structures by hand (or retrieve the data multiple times from a database).

sort_expression
An optional **sortBy** by expression may also be given. In this case, the expression will be evaluated for each row of the query given, and then the result set will be sorted in ascending order by the results of the expressions according to the resulting type of the evaluated expression (i.e. if the result of the evaluation of the expression gives a string, then string order is used to sort, if the result of the evaluation is an integer, then integer order is used, etc).

sort_descending_expression
Another optional modifier to the **context** statement that behaves the same as above except that the results are sorted in descending order.

Example

```
# note that "%service_type" and "%effective_start_date" represent values
# in the $service_history hash of arrays.
context ($service_history) where (%service_type == "voice")
```

```
sortBy (%effective_start_date) {
   printf("%s: start date: %s\n", %msisdn, format_date("YYYY-MM-DD HH:mm:SS", %
      effective_start_date));
}
```

See also

- Context Functions
- Find Expressions
- subcontext Statements
- summarize Statements
- HashListIterator
- ListHashIterator
- Qore::zzz8hashzzz9::contextIterator()

23.14 summarize Statements

Synopsis

summarize statements are like context statements with one important difference: results sets are grouped by a by expression, and the statement is executed only once per discrete by expression result. This statement is designed to be used with the subcontext statement.

Syntax

summarize (*data_expression*) **by** (*by_expression*)
[**where** (*where_expression*)]
[**sortBy** (*sort_expression*)]
[**sortDescendingBy** (*sort_descending_expression*)]
statement

Description

summarize statements modifiers have the same effect as those for the context statement, except for the following:

by (*by_expression*)
The **by** expression is executed for each row in the data structure indicated. The set of unique results defines groups of result rows. For each group of result rows, each row having an identical result of the evaluation of the by expression, the statement is executed only once.

Example

```
# note that "%service_type" and "%effective_start_date" represent values
# in the $services hash of arrays.
summarize ($services)
    by (%effective_start_date)
    where (%service_type == "voice")
    sortBy (%effective_start_date) {
    printf("account has %d service(s) starting on %s\n",
           context_rows(),
           format_date("YYYY-MM-DD HH:mm:SS", %effective_start_date));
}
```

See also

- Context Functions
- Find Expressions
- context Statements
- subcontext Statements
- HashListIterator
- ListHashIterator

23.15 subcontext Statements

Synopsis

Statement used to loop through values within a summarize statement.

Syntax

subcontext
[**where** (*where_expression*)]
[**sortBy** (*sort_expression*)]
[**sortDescendingBy** (*sort_descending_expression*)]
statement

Description

The **subcontext** statement is used in conjunction with summarize statements. When result rows of a query should be grouped, and then each row in the result set should be individually processed, the Qore programmer should first use a summarize statement, and then a **subcontext** statement. The summarize statement will group rows, and then the nested **subcontext** statement will iterate through each row in the current summary group.

Example

```
summarize ($services)
    by (%effective_start_date)
    where (%service_type == "voice")
    sortBy (%effective_start_date) {
    printf("account has %d service(s) starting on %s\n",
           context_rows(),
           format_date("YYYY-MM-DD HH:mm:SS", %effective_start_date));
    subcontext sortDescendingBy (%effective_end_date) {
        printf("\tservice %s: ends: %s\n", %msisdn, format_date("YYYY-MM-DD HH:mm:SS", %
      effective_end_date));
    }
}
```

See also

- Context Functions
- Find Expressions
- context Statements
- summarize Statements
- HashListIterator
- ListHashIterator

23.16 return Statements

Synopsis

return statements causes the flow of execution of the function, method or program to stop immediately and return to the caller. This statement can take an optional expression to return a value to the caller as well.

Syntax

return *[expression]*;

Description

This statement causes execution of the current function, method, or program to returns to the caller, optionally with a return value.

Example

```
string sub getName() {
   return "Barney";
}
my string $name = getName();
```

23.17 on_exit Statements

Synopsis

Queues a statement or statement block for unconditional execution when the block is exited, even in the case of exceptions or return statements. For similar statement that queue code for execution depending on the exception status when the block exits, see on_success statements and on_error statements.

Syntax

on_exit
statement

Description

The **on_exit** statement provides a clean way to do exception-safe cleanup within Qore code. Any single statment (or statement block) after the **on_exit** keyword will be executed when the current block exits (as long as the statement itself is reached when executing - **on_exit** statements that are never reached when executing will have no effect).

The position of the **on_exit** statement in the block is important, as the immediate effect of this statement is to queue its code for execution when the block is exited. Even if an exception is raised or a return statement is executed, any **on_exit** code that is queued will be executed. Therefore it's ideal for putting cleanup code right next to the code that requires the cleanup.

Note that if this statement is reached when executing in a loop, the **on_exit** code will be executed for each iteration of the loop.

By using this statement, programmers ensure that necessary cleanup will be performed regardless of the exit status of the block (exception, return, etc).

Example

```
{
    $mutex.lock();
    # here we queue the unlock of the mutex when the block exits, even if an exception is thrown below
    on_exit $mutex.unlock();
    if ($error)
        throw "ERROR", "Scary error happened";
    print("everything's OK!\n");
    return "OK";
}
# when the block exits for any reason, the mutex will be unlocked
```

23.18 on_success Statements

Synopsis

Queues a statement or statement block for execution when the block is exited in the case that no exception is active. Used often in conjunction with the on_error statement and related to the on_exit statement.

Syntax

on_success
statement

Description

The **on_success** statement provides a clean way to do block-level cleanup within Qore code in the case that no exception is thrown in the block. Any single statment (or statement block) after the **on_success** keyword will be executed when the current block exits as long as no unhandled exception has been thrown (and as long as the statement itself is reached when executing - **on_success** statements that are never reached when executing will have no effect).

The position of the **on_success** statement in the block is important, as the immediate effect of this statement is to queue its code for conditional execution when the block is exited. Even if a return statement is executed later in the block, any **on_success** code that is queued will be executed as long as there is no active (unhandled) exception. Therefore it's ideal for putting cleanup code right next to the code that requires the cleanup, along with on_error statements, which are executed in a manner similar to **on_success** statements, except **on_error** statements are only executed when there is an active exception when the block is exited.

Note that if this statement is reached when executing in a loop, the **on_success** code will be executed for each iteration of the loop (as long as there is no active exception).

Example

```
{
    $db.beginTransaction();
    # here we queue the commit in the case there are no errors
    on_success $db.commit();
    # here we queue a rollback in the case of an exception
    on_error $db.rollback();
    $db.select("select * from table where id = %v for update", $id);
    # .. more code

    return "OK";
}
# when the block exits. the transaction will be either committed or rolled back,
# depending on if an exception was raised or not
```

23.19 on_error Statements

Synopsis

Queues a statement or statement block for execution when the block is exited in the case that no exception is active. Used often in conjunction with the on_success statement and related to the on_exit statement.

Syntax

on_error
statement

Description

The **on_error** statement provides a clean way to do block-level cleanup within Qore code in the case that an exception is thrown in the block. Any single statment (or statement block) after the **on_error** keyword will be executed when the current block exits as long as an unhandled exception has been thrown (and as long as the statement itself is reached when executing - **on_error** statements that are never reached when executing will have no effect).

The position of the **on_error** statement in the block is important, as the immediate effect of this statement is to queue its code for conditional execution when the block is exited. Even if a return statement is executed later in the block, any **on_error** code that is queued will be executed as long as there is an active (unhandled) exception. Therefore it's ideal for putting cleanup code right next to the code that requires the cleanup, along with on_success statements, which are executed in a manner similar to **on_error** statements, except on_success statements are only executed when there is no active exception when the block is exited.

Note that the code in this statement can only be executed once in any block, as a block (even a block within a loop) can only exit the loop once with an active exception (in contrast to on_success and on_exit statements, which are executed for every iteration of a loop).

Example

```
 {
    $db.beginTransaction();
    # here we queue the commit in the case there are no errors
    on_success $db.commit();
    # here we queue a rollback in the case of an exception
    on_error $db.rollback();
    $db.select("select * from table where id = %v for update", $id);
    # .. more code

    return "OK";
}
# when the block exits. the transaction will be either committed or rolled back,
# depending on if an exception was raised or not
```

Chapter 24

Functions

24.1 Function Declarations

A function is declared in Qore by using the keyword **sub** (for subroutine) as follows.

Function Declaration Syntax

```
[public] [synchronized] [deprecated] [return_type] sub function_name ([param_type] $var_name ...]) {
statement(s)...
}
```

or the deprecated alternate syntax with the **returns** keyword:

```
[public] [synchronized] [deprecated] sub function_name ([param_type] $var_name ...])
returns return_type {
statement(s)...
}
```

Function names must be valid Qore identifiers.

When defining a user module, the function definition can be preceded by public, which means that the function variant will be available (imported) in the Program object importing the module. See public for more information.

24.1.1 Function Parameters

Variables listed in parentheses after the function name are the parameters to the function and automatically get local lexical scoping.

Type declarations optionally precede the parameter variable and will restrict any arguments passed to the type declared. The same function can be declared multiple times if each declaration has different parameter types; this is called overloading the function.

Variables passed as function arguments are passed by value by default, unless the caller places a `"\"` character before an lvalue in the argument list in the function call. In this case the function must have a parameter defined to accept the variable passed by reference. Any changes to the local variable will be reflected in the original variable for variables passed by reference. Also note that it is illegal to pass a local variable by reference in a background expression.

Note

In order to process a variable number of arguments to a function, use implicit argument references (`$1`) or the `$argv` variable (an automatic local variable); these are automatically instantiated at run time when additional arguments in excess of the declared parameters are passed to the function at run time. No declaration needs to be made in the function signature to use the `$argv` variable.

24.1.2 Function Return Type Declarations

The return type of the function can be given by placing a type declaration before the **sub** keyword (the older syntax with the **returns** keyword after the parameter list is still accepted as well).

Note

Parameter and return types are required when the Qore::PO_REQUIRE_TYPES or Qore::PO_REQUIRE_↩PROTOTYPES parse options are set.

Functions use the return statement to provide a return value to the caller.

24.1.3 Simple Example Functions

Here is an example function declaration returning a value:

```
#!/usr/bin/qore
# function declaration example

int sub print_string(string $string) {
    print("%s\n", $string);
    return 1;
}
```

Functions may also be recursive. Here is an example of a recursive Qore function implementing the Fibonacci function:

```
#!/usr/bin/qore
#
# recursive function example

int sub fibonacci(int $num) {
    if ($num == 1)
        return 1;
    return $num * fibonacci($num - 1);
}
```

Note

Function names are resolved during the second parse pass; therefore functions do not need to be declared before being referenced. This allows an easy definition of 2 or more self-referencing functions.

24.1.4 "Synchronized" Functions

Functions declared with the **synchronized** keyword will only run in one thread at a time.

If another thread tries to call the function while the function is already being executed, any callers will block until the function has finished executing.

Note that the lock used is recursive, so that a single thread may call the function multiple times safely without fear of a deadlock. The lock used also participates in Qore's deadlock detection framework, therefore if a deadlock is detected, a `THREAD-DEADLOCK` exception is thrown.

See also

synchronized

24.1.5 "Deprecated" Functions

Functions declared with the **deprecated** keyword will cause a deprecated warning to be raised when the function is referenced (but not when it's declared).

In this way API functions (or methods) can be declared as **deprecated** before eventual removal from the API set.

Chapter 25

Code Flags

Builtin functions (and methods) can be tagged with flags that give some properties of the code to be executed as in the following sections.

25.1 NOOP

Code with this flag makes no calculations, but rather returns a constant value. This flag is given to function and method variants that return a default value depending on the type of argument(s). When variants with this flag are resolved at parse time, a call-with-type-errors warning is raised (assuming this warning is enabled).

25.2 RUNTIME_NOOP

Code with this flag makes no calculations, but rather returns a constant value. This flag is given to function and method variants that return a default value depending on the type of argument(s). When variants with this flag are resolved at parse time, a call-with-type-errors warning is raised (assuming this warning is enabled), unless Qore::PO_REQUIRE_TYPES or Qore::PO_STRICT_ARGS is set. If either Qore::PO_REQUIRE_TYPES or Qore::PO_STRICT_ARGS is set, then these variants are inaccessible; resolving to a variant with this flag set at parse time or run time causes an exception to be thrown. These variants are included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

25.3 CONSTANT

This flag indicates that the function or method has no side effects and does not throw any exceptions (see also RET_VALUE_ONLY).

25.4 RET_VALUE_ONLY

This flag indicates that the function or method has no side effects but could throw an exception (see also CONSTANT).

25.5 DEPRECATED

Code with this flag is deprecated and may be removed in a future version of Qore; if a variant with this flag is resolved at parse time, a "Deprecated" Functions warning is raised (assuming this warning is enabled).

Chapter 26

Namespaces

Description

Namespaces allow constants, classes, functions, global variables, and even other namespaces to co-exist in the same program by defining them in separate namespaces. Constants, classes, functions, and sub-namespaces can be declared to belong to a particular namespace either by defining them in-line within a namespace declaration, or by including the namespace name/path prepended to the constant, class, function, or namespace declaration separated by two colons **"::"**. Global variables can only be declared out of line so they can be initialized when they are declared.

If the user does not specify the parent namespace with a namespace path in constant, class, function, global variable, or namespace declarations, the declaration will be by default in the unnamed default root namespace.

In-Line Namespace Declaration

```
[public] namespace [namespace_path::]namespace_identifier {
[constant_declarations]
[class_declarations]
[function_declarations]
[global_variable_declarations]
[namespace_declarations]
}
```

Out of Line Namespace Declaration

```
[public] namespace namespace_identifier;
```

Note

No namespace path may be given with out of line namespace declarations.

When defining a user module, namespace declarations can be preceded by public, which means that the namespace's public contents will be exported into the Program object importing the module. When a namespace is declared public outside of a user module, it means that the namespace can be inherited in any child Program objects created in the same scope. S ee public for more information.

Namespace Resolution

Namespaces can either be resolved by giving a path to the constant, class, function, global variable, or namespace desired, or by leaving out the namespace path and allowing the system to search for the constant, class, function, global variable, or namespace. In either case, a depth-first search of the namespace tree is made for a match.

If a namespace path is included, then the tree is searched for the first namespace match, and, if the rest of the declaration cannot be matched, the search continues in the entire namespace tree until a complete match is found.

Namespace Paths

Namespace paths look like the following:

- *starting_namespace::[sub_namespace(s)s::]constant|class|namespace*

Example

```
namespace MyNamspace {
    # global variable declarations in a namespace declaration cannot be initialized at the point they are
       declared
    our bool $my_bool;

    const MyConst = 1;
    class MyClass {
    }
    namespace MySubNamespace;
}

# out of line initialization for MyNamespace::$my_bool
MyNamespace::$my_bool = False;
# out of line declaration and initialization for MyNamespace::$my_int
our int MyNamespace::$my_int = 200;
```

Note

No semicolon (";") should follow the closing bracket in a namespace declaration (in fact using a semicolon in this case would raise a parse exception).

Chapter 27

Constants

Description

Constant definitions allow programmers to refer to values with Qore identifiers rather than using the value or the expression that generates the value.

Constant Declaration Syntax

[public] **const** *[namespace_path::]constant_identifier* = *initialization_expression*;

Example

```
const PI_SQUARED = pow(Qore::M_PI, 2);
```

When defining a user module, constant definitions can be preceded by public, which means that the constant will be available (imported) in the Program object importing the module. Note that however the public keyword cannot be used when defining class constants in a public or private declaration block. See public for more information.

Since

0.8.1 The expression used to initialize a constant can be any valid Qore expression as long as no variables are referenced.

Note

It is bad programming practice to assign a constant with an expression that has side effects. Furthermore if an expression used to initialize a constant throws an exception, that exception cannot be caught in the program being defined.

The order that constants are initialized and assigned does not necessarily correspond to declaration order, however constants may be defined using the values of other constants; if a circular reference is detected a parse exception is raised.

Constants are resolved in the second stage of parsing (during the parse commit stage, see also Program::parseCommit()

Classes can also declare constants; see class_constants for more information.

Chapter 28

Classes

28.1 Class Overview

Classes define types of Qore objects. Classes can define members and methods, which are attributes of the class and functions that operate only on the objects of that class, respectively. Furthermore access to class members and methods can be restricted with the **private** keyword, and classes can be subclassed to support polymorphism.

Note

Each Qore type has a "pseudo-class" associated with it (the default is Qore::zzz8valuezzz9); methods from the data type's "pseudo-class" can be run on any value of that type; see Qore::zzz8valuezzz9 and **"Pseudo Class for Type"** headings in Basic Data Types for more information.

In-Line Class Declaration Syntax

```
[public] [final] class [namespace_path::...]class_identifier] [inherits [private|public]
[namespace_path::...]parent_class_identifier[, ...]] {
[[private] $.member_name[, ...];]

[private|public {
[[member_type] $.member_name [= initialization_expression];]
[static [static_member_type] static_member_name [= initialization_expression];]
[const constant_name = initialization_expression;]
}]

[[final] [private|public] [deprecated] constructor([[param_type] $param_name
[= default_initialization_expression], ...]) [: parent_class_name(args...), ...] {
}]
[[static|final] [synchronized] [private|public] [deprecated] method_name([[param_type] $param_name [= default_initialization_expression], ...]) {
}]
[[abstract] [private|public] method_name([[param_type] $param_name [= default_initialization_expression], ...]);
...
}
```

Note

To define a class interface, define a class with abstract methods.

When defining a user module, class declarations can be preceded by public, which means that the class will be available (imported) in the Program object importing the module. When a class is declared public outside of a user module, it means that the class can be inherited in any child Program objects created in the same scope. See public for more information. Note that classes can also be imported singly by using the Program::importClass() method.

In-Line Class Declaration Example

```
class MyClass inherits MyBaseClass {
    public {
        string $.pub_attr = "hello";
        const pub_const = "foo";
        static pub_static_var = "bar";
    }

    constructor(string $arg) : MyBaseClass($arg + 1) {
    }

    softstring myMethod(softint $i) {
        return $i + 2;
    }
}
```

Note

No semicolon (";") is required to terminate a class declaration (in fact using a semicolon would raise a parse exception).

Out-Of-Line Class Declaration Syntax

Alternatively class declarations and method declarations can be defined out of line as follows:
[public] **class** [*namespace_path*::...] *class_identifier* [**inherits** [**private**|**public**] [*namespace_path*::...] *parent_class_identifier* [, ...]];

[[**private**|**public**] [**deprecated**] [*namespace_path*::...] *class_identifier*::**constructor**([*param_type*] $*param_name* [= *default_initialization_expression*], ...]) [: *parent_class_name*(*args...*), ...] {
}]

[[**static**|**final**] [**synchronized**] [**private**|**public**] [**deprecated**] [*namespace_path*::...] *class_identifier*::*method_name*([*param_type*] $*param_name* [= *default_initialization_expression*], ...]) {
}] [**abstract** [**private**|**public**] [*namespace_path*::...] *class_identifier*::*method_name*([*param_type*] $*param_name* [= *default_initialization_expression*], ...]);
As with inline class definitions, when defining a user module, class declarations can be preceded by public, which means that the class will be available (imported) in the Program object importing the module. When a class is declared public outside of a user module, it means that the class can be inherited in any child Program objects created in the same scope. See public for more information. Note that classes can also be imported singly by using the Program::importClass() method.

Out-Of-Line Class Declaration Example

```
class MyNamespace::MyClass inherits MyBaseClass;

MyNamespace::MyClass::constructor(string $arg = "temp") : MyBaseClass($arg + ".txt") {
}

softstring MyNamespace::MyClass::myMethod(softint $i) {
    return $i + 2;
}
```

Note

Because method definitions are allowed out-of-line, this means that builtin classes may be extended with new user methods, however user constructor, destructor, and copy methods cannot be added to builtin classes; to customize the behavior of these methods for builtin classes, subclass the class instead.

When parse option %allow-bare-refs is enabled, no "$" or "$." prefixes can be used with variable or method or member names as in the specifications above. Class members, class constants, and static class variables can only be declared in an in-line class definition (the first example above). If a class has at least one public member declared (or inherits a class with at least one public member declared), then only those members declared as public can be accessed from outside the class, and from within the class only members

explicitly declared can be accessed as well (unless the class also defines a memberGate() method). In this way typographical errors in member names can be caught (at parse time if types are declared).

In a class hierarchy, base class constructor methods can be explicitly specified using a special syntax unique to subclass constructor methods. Please see Class Inheritance for more information.

It's possible to write purely object-oriented scripts/programs in Qore by defining an application class and using the "`-x`" or "`--exec-class`" command-line arguments to tell Qore to instantiate the class instead of doing normal top-level execution (in fact, the "`--exec-class`" argument disallows the use of top-level statements entirely). For more information, please see qore Executable Command-Line Processing and Parse Directives.

Final Classes

Classes declared **`final`** cannot be subclassed, and, by extension, no methods of a **`final`** class can be reimplemented as well.

Currently the Qore standard class library (delivered with Qore) has no **`final`** classes.

28.2 Class Methods

Public Methods

All class methods are public by default, but methods can also be explicitly declared with the **`public`** keyword as well. Public methods have no class protection and can be called from any code context.

Private Methods

Methods declared with the **`private`** keyword can only be called by other member functions of the same class or of derived classes. Any attempt to call these methods from outside the class hierarchy will result in a run-time exception.

Synchronized Methods

Methods declared with the **`synchronized`** keyword will only run in one thread at a time.

For more information, see the synchronized keyword.

Static Methods

Methods declared with the **`static`** keyword are like regular functions that are attached to the class. These methods are not associated with a particular object's state and therefore are not allowed to refer to object members or call non-static methods. Also, no reference to the special `$self` variable is allowed within static methods.

Static methods may be declared **`private`** or **`public`** or **`synchronized`** like non-static methods; static methods can also access private members of a class (through an object of the class for non-static members).

Static method calls take a special syntax as documented in Static Method Calls.

Abstract Methods

Methods declared with the **`abstract`** keyword define methods interfaces that must be defined in child classes for the class to be instantiated. Methods defining the declared interfaces in child classes must define exactly the same paramters in order for the abstract method to match, but the return type of the concrete method in the child class has to simply be compatible with the return type of the abstract method in the parent class.

Classes with undefined abstract methods cannot be instantiated. Abstract methods cannot have a method body; an abstract method's declaration must be terminated with a semicolon (`";"`).

The **`abstract`** keyword can only be used in abstract method declaration; the use of this keyword with a method declaration with a method body will cause a parse exception to be thrown.

Constructors, Destructors, and Other Special Methods

All class methods are optional, but some methods have a special meaning.

Special Methods

Name	Description
`constructor(`*[params...]*`) {}`	Called when objects are created when instantiated by a variable declaration with a class type and constructor arguments or explicitly with the new operator. User code may not explicitly call `constructor()` methods directly. In a class tree, `constructor()` methods are called for base classes first in left-to-right, depth-first declaration order. `constructor()` methods may be overloaded and also private constructors may be defined. Private constructors can only be called from within the class.
`copy() {}`	When a user explicitly calls a copy method, Qore will generate a new object with references to the same members as the source object. Then, if there are any base classes, base class `copy()` methods are called in the same order as the `constructor()` methods. If a `copy()` method is defined, it will be run in the new object with a reference to the old object passed as the first argument. Any other arguments passed to the `copy()` method are ignored. `copy()` methods cannot be overloaded and cannot be private.

`destructor() {}`	Called when objects go out of scope or are explicitly deleted. User code may not explicitly call `destructor()` methods. In a class tree, `destructor()` methods are called for base classes in the opposite order in which the constructors are called. `destructor()` methods cannot be overloaded and cannot be private.
any `memberGate` (string *$member_param_name*) `{}`	If this method is implemented in the class, it is called when read access is attempted to private member or members that do not exist in the current object; the return value of this method is returned as the value of the member. `memberGate()` methods cannot be overloaded and are not inherited by subclasses.
any `methodGate` (string *$method_param_name*, `...`) `{}`	If this method is implemented in the class, it is called when methods are called on the object that do not exist in the current object and the return value of this method is returned as the value of the method call. `methodGate()` methods cannot be overloaded and are not inherited by subclasses.
`memberNotification` (string *$member_param_name*) `{}`	If this method is implemented in the class, it is called when an object member is updated outside the class with the member name as the argument. Note that this method is called after the member has been updated and without locking; the call is not atomic respective to other threads that also may update the same member simultaneously. `memberNotification()` methods cannot be overloaded and are not inherited by subclasses.

Final Methods

Methods declared **`final`** cannot be reimplemented in a subclass. Methods can be declared **`final`** to prevent a subclass from altering the behavior of a method that is critical to the class's functionality.

None of the following special methods can be declared **`final`**:

- constructor(): declare the class final instead
- destructor(): declare the class final instead
- copy(): declare the class final instead
- memberGate(): these methods are not inherited so they may not be declared **`final`**
- memberNotification(): these methods are not inherited so they may not be declared **`final`**
- methodGate(): these methods are not inherited so they may not be declared **`final`**

Currently the Qore standard class library has no **`final`** methods.

Class Constants

Class constants, like non-class constants, allow programmers to refer to values with Qore identifiers rather than using the value or the expression that generates the value.

See Class Overview for a description of the syntax required to declare a class constant.

Like other attributes of classes, class constants may be declared **private** or **public**. The following is an example of a class constant definition:

```
class Test {
    public {
        const Version = "1.0";
    }

    private {
        const Limit = 100;
    }
}
```

Note

Class constants cannot have the same names as static class variables.

Static Class Variables

Static class variables are like global variables that belong to a class. They are not associated with any particular object.

See Class Overview for a description of the syntax required to declare a static class variable.

Like other attributes of classes, static class variables may be declared **private** or **public**. The following is an example of a static class variable definition:

```
class Test {
    public {
        static string lastFile = "none";
    }

    private {
        static int numProcessed = 0;
    }
}
```

Note

Static class variables cannot have the same same as class constants.

28.3 Class Members

Public Member Declarations

If a class has at least one public member declared (or inherits a class with at least one public member declared), then only those members declared as public can be accessed from outside the class, and from within the class only members explicitly declared can be accessed as well (unless the class also defines a memberGate() method). In this way typographical errors in member names can be caught (at parse time if types are declared).

Private Members

Members declared **pivate** can only be accessed within the class hierarchy; trying to access private members form outside the class hierarchy will result in either a parse or runtime exception, depending on when the illegal acceess is caught.

Class Member References

When defining a class, members of instantiated objects are referred to with a special syntax as follows:
$. *member_name*

Furthermore, the automatic variable `$self` is instantiated in every non-static method, representing the current object (similar to `this` in C++ or Java). Therefore if you need to access hash members which are not valid Qore identifiers, then enclose the member name in double quotes after the dot operator as follows:

```
$self."&member-name"
```

memberGate() Method

If the class implements a `memberGate()` method, then whenever a non-existent member of the class is accessed (read), this method will be called with the name of the member as the sole argument, so that the class can create the member (or react in some other way) on demand. This method is also called when methods of the same class try to access (read) non-existent methods, but is not called from within the `memberGate()` method itself.

memberNotification() Methods

To monitor writes to the object, the class can implement a `memberNotification()` method, which is called whenever an object member is modified from outside class member code. In this case, the `memberNotification()` method is called with the name of the member that was updated so that an object can automatically react to changes to its members (writes to members) from outside the class. This method is not called when members are updated from within class member code.

Member Initialization

Members that have intialization expressions in the class definition are initialized before the constructor is executed (but after any base class constructors have run). An exception raised in a member initialization expression will cause the constructor to fail and for the object to be deleted immediately.

Note

The automatic `$argv` local variable is instantiated as usual in all class methods where there are more arguments than variables declared in the method declaration.

28.4 Object Method Calls

In-Class Method Call Syntax

Within a class method definition, calls to methods in the same class hierarchy (of the current class or a base class) can be made as follows:
`[[`*namespace_path*`:::...]`*parent_class_name*`::]$.`*method_name*`([`*args*`, ...])`

In-Class Method Call Example

```
# to call a specific method in a base class
Thread::Mutex::$.lock();
# to call lock() in the current (or lower base) class
$.lock();
```

In-Class Method Calls with allow-bare-refs

When the %allow-bare-refs parse directive is set, then object methods are called without the `"$."` prefix as in the following example:

```
# to call a specific method in a base class
Thread::Mutex::lock();
# to call lock() in the current (or lower base) class
lock();
```

Calls to object methods can be made outside the class by using the above syntax as well. If the object's class is not known at parse time, then the call is resolved at run-time, and if a call is attempted to a private function outside the defining class, then a run-time `METHOD-IS-PRIVATE` (if the method is private) or `BASE-CLASS-IS-PRIVATE` (if the method resolves to a privately-inherited base class) exception is raised.

methodGate() methods

If the class implements a `methodGate()` method, then whenever a non-existent method of the class is called, the methodGate() method will be called with the name of the member as the first argument (prepended to the other arguments to the non-existant method), so that the class simulate or redirect the method call. This method is also called when methods of the same class try to call non-existent methods, but is not called from within the `methodGate()` method itself.

28.5 Class Inheritance

Class inheritance is a powerful concept for easily extending and resuing object-oriented code, but is also subject to some limitations. This section explains how class inheritance works in Qore.

Classes inherit the methods of a parent class by using the **`inherits`** keyword as specified above. Multiple inheritance is supported; a single Qore class can inherit one or more classes. When a class is inherited by another class, it is called a base class or parent class. Private inheritance is specified by including the keyword **`private`** before the inherited class's name. When a class is privately inherited, it means that the inherited class's public declarations (members, constants, methods, etc) are treated as private in the context of accesses outside the class.

Inheritance is **`public`** by default, to inherit a class privately, use the **`private`** keyword before the class name or class path to inherit as follows:

```
class ChildClass inherits private ParentClass {
}
```

It is not legal to directly inherit the same class directly more than once; that is; it is not legal to list the same class more than once after the **`inherits`** keyword. However, it is possible that a base class could appear more than once in the inheritance tree if that class is inherited separately by two or more classes in the tree.

In this case, the base class will actually only be inherited once in the subclass, even though it appears in the inheritance tree more than once. This must be taken into consideration when designing class hierarchies, particularly if base class constructor parameters for that class are explicitly provided in a different way by the inheriting classes.

Note

Class members only exist once for each object; therefore if classes in an inheritance tree have different uses for members with the same name, then a class hierarchy built of such classes will probably not function properly.

Subclasses can give explicit arguments to their base class constructors using a special syntax (only available to subclass constructors) similar to the C++ syntax for the same purpose as follows:
`[[`**`private`**`|`**`public`**`]` `[`**`deprecated`**`]` **`constructor`**`(`*`[param_type]`* `$`*`param_name`* `[=` *`default_initialization_expression`*`], ...]) [:` *`parent_class_name`*`(`*`args...`*`), ...]` `{ }]`

Here is a concrete example of giving arguments to an inherited base class:

```
class XmlRpcClient inherits Qore::HTTPClient {
    # calls the base class HTTPClient constructor, overrides the "protocols" key to "xmlrpc"
    constructor(hash $opts = hash()) : Qore::HTTPClient($opts + ( "protocols" : "xmlrpc" )) {
    }
}
```

Because base class constructors are executed before subclass constructors, the only local variables in the constructor that can be referenced are those declared in the subclass constructor declaration (if any). What this means

is that if you declare local variables in the expressions giving base class arguments, these local variables are not accessible from the constructor body.

Note

Base classes that give explicit arguments to their base class constructors can be overridden by subclasses by simply listing the base class in the base class constructor list and providing new arguments.

28.5.1 Private Inheritance

Classes inherited using the **`private`** keyword encapsulate the functionality (members, constants, methods, etc) of the parent class privately in the child class; that is, any access to the privately-inherited parent class's functionality from outside the class will result in either a parse-time or runtime exception, depending on when the error is caught.

28.5.2 Public Inheritance

Classes are inherited publically by default; public inheritance means that the parent class's functionality (members, constants, methods, etc) have the same visibility in the child class as they do in the parent class. For example, a public method in the parent class will also be public in the child class if the parent class is inherited publically. Respectively, private methods of publically-inherited parent classes will still be private in child classes.

Chapter 29

Threading

A thread is an independent sequence of execution of Qore code within a Qore program or script. Each thread has a thread ID or TID.

The first thread of execution in a Qore program has TID 1. TID 0 is always reserved for the special signal handler thread.

The Qore language is designed to be thread-safe and Qore programs should not crash the Qore executable due to threading errors. Threading errors should only cause exceptions to be thrown or application errors to occur.

Threading functionality in Qore is provided by the operating system's POSIX threads library.

29.1 Creating and Terminating Threads

New threads are created with the background operator. This operator executes the expression given as an argument in a new thread and returns the TID (integer thread ID) of the new thread to the calling thread. This is most useful for calling user functions or object methods designed to run in a separate thread.

To terminate a thread, the thread_exit statement should be called, as calling the exit() function will terminate the entire process (and therefore all threads) immediately.

29.2 Threading and Variables

All global variables are shared in Qore programs, while local variables (declared with **my**) are generally local to each thread (and thus accessed without any mutual-exclusion locking), regardless of location. This means that if a variable is declared with **my** at the top level, it will actually have global scope, but also each thread will have its own copy of the variable. In effect, declaring a top-level local variable with **my** actually creates a global thread-local variable.

The following code gives an example of declaring a global thread-local variable by using **my** at the top-level:

```
%require-our
sub t() {
    printf("x=%y\n", $x);
}
my int $x = 2;
t();
background t();
```

This will print out:

```
x=2
x=null
```

Note that the second time the local variable is accessed in the background thread, it has no value.

Due to the way Qore's local variables work, it is illegal to declare a top-level local variable after first block is parsed in the program; that is; if any call to parse() or Qore::Program::parse() is made in an existing program (where a top-level block already exists), and an attempt to declare a new top-level local variable is made, then a `ILLEGAL-TOP-LEVEL-LOCAL-VARIABLE` parse exception will be raised.

Access to global variables in qore is wrapped in mutual-exclusion locks to guarantee safe access to global variable data in a multithreaded context. Local variables are thread-local and therefore not locked, except when referenced in a closure or when a reference is taken of them, in which case the local variable's scope is extended to that of the closure's or the reference's, and all accesses to the bound local variable are made within mutual-exclusion locks as these variables may be used in multithreaded contexts.

An alternative to global thread-local variables is offered by the save_thread_data() and get_thread_data() functions (documented in Threading Functions).

29.3 Thread Synchronization and Inter-Thread Communication

synchronized

The **`synchronized`** keyword can be used before function or class method definitions in order to guarantee that the function or method call will only be executed in one thread at a time. As in Java, this keyword can also be used safely with recursive functions and methods (internally a recursive mutual exclusion lock that participates in Qore's deadlock detection framework is used to guarantee thread-exclusivity and allow recursion).

Classes Useful With Threading

The following classes are useful when developing multi-threaded Qore programs:

Class	Description
Mutex	A mutual-exclusion thread lock
Gate	A recursive thread lock
RWLock	A read-write thread lock
Condition	Allows Qore programs to block until a certain condition becomes true
Counter	A blocking counter class
Queue	A thread-safe, blocking queue class (useful for message passing)
Sequence	A simple, thread-atomic sequence object (increment-only)
ThreadPool	A flexible, dynamically scalable thread pool
AutoLock	A helper class to automatically release Mutex locks when the AutoLock object is deleted
AutoGate	A helper class to automatically exit Gate locks when the AutoGate object is deleted
AutoReadLock	A helper class to automatically release read locks when the AutoReadLock object is deleted
AutoWriteLock	A helper class to automatically release read locks when the AutoWriteLock object is deleted

Functions Useful With Threading

The following functions assist writing safe and efficient multi-threaded Qore programs:

Function	Description
save_thread_data()	Saves a thread-local value against a key.
get_all_thread_data()	Retrieves the entire thread-local hash.

get_thread_data()	Retrieves a thread-local value based on a key.
delete_all_thread_data()	Deletes the entire thread-local data hash.
delete_thread_data()	Delete the value of a key in the thread-local data hash.
gettid()	Gets the thread's TID (thread identifier)
thread_list()	Returns a list of TIDs of running threads
num_threads()	Returns the number of running threads
throwThreadResourceExceptions()	runs thread-resource cleanup routines and throws the associated exceptions
mark_thread_resources()	sets a checkpoint for throwing thread resource exceptions
throw_thread_resource_exceptions_to_mark()	runs thread-resource cleanup routines and throws the associated exceptions to the last mark and clears the mark

29.4 Deadlocks

Qore supports deadlock detection in complex locking scenarios and will throw a `THREAD-DEADLOCK` exception rather than allow an operation to be performed that would cause a deadlock. Deadlock detection is implemented for internal locking (global variable and object access), synchronized methods and functions, etc, as well as for all Qore threading classes.

Qore can only detect deadlocks when a lock resource acquired by one thread is required by another who holds a lock that the first thread also needs. Other errors such as forgetting to unlock a global lock and trying to acquire that lock in another thread cannot be differentiated from valid use of threading primitives and will result in a process that never terminates (a deadlocked process). However, common threading errors such as trying to lock the same Mutex twice in the same thread without unlocking it between the two Mutex::lock() calls are caught in Qore and exceptions are thrown. Additionally, locks are tracked as thread resources, so if a thread terminates while holding a lock, an exception will be thrown and the lock will be automatically released.

Chapter 30

Exception Handling

Exceptions are errors that can only be handled using a try catch block. Any exception that is thrown in a try block will immediately cause execution of that thread to begin with the first statement of the catch block, regardless of the position of the program pointer of the running thread, even if nested function or object method calls have been made.

Exceptions can be thrown by the Qore system for a number of reasons, see the documentation for each function and object method for details.

Programmers can also throw exceptions explicitly by using the throw and rethrow statements.

Information about the exception, including the context in which the exception occurred, is saved in the exception hash, which can be retrieved by using a parameter variable in the catch block.

The exception hash contains the following members:

Exception Hash Keys

Name	Type	Description
`type`	string	see Exception Type Constants for possible values
`file`	string	The parse label where exception occurred; this is normally the file name; this corresponds to the *label* parameter of the Program::parse() and Program::parsePending() methods and the Qore::parse() function, for example
`line`	int	The starting line number where exception occurred
`endline`	int	The ending line number where the exception occurred
`source`	*string	An optional source string for the exception; if multiple sections of a file were parsed with different parse labels, then the source file name will normally go here and the `"file"` key will have the parse label; this corresponds to the *source* parameter of the Program::parse() and Program::parsePending() methods, for example

offset	int	The line number offset for the "source" key
callStack	list of hashes	Backtrace information
err	any	This key is populated with the value of the first expression of the throw statement. For system exceptions, this is a string giving the exception code.
desc	any	This key is populated with the value of the second expression of the throw statement (if a list was thrown). For system exceptions, this is a string giving a text description of the error.
arg	any	This key is populated with the value of the third expression of the throw statement (if a list was thrown). For system exceptions, this is populated for some exceptions where additional information is provided.

Call Stack Description

Name	Type	Description
function	string	function name of the source where the exception was raised (if known)
file	string	The parse label where exception occurred; this is normally the file name; this corresponds to the *label* parameter of the Program::parse() and Program::parsePending() methods and the Qore::parse() function, for example (if known, for user exceptions only)
line	int	The starting line number where exception occurred (if known, for user exceptions only)
endline	int	The ending line number where the exception occurred (if known, for user exceptions only)
source	*string	An optional source string for the exception; if multiple sections of a file were parsed with different parse labels, then the source file name will normally go here and the "file" key will have the parse label; this corresponds to the *source* parameter of the Program::parse() and Program::parsePending() methods, for example (if known, for user exceptions only)

`offset`	int	The line number offset for the `"source"` key (if known, for user exceptions only)
`type`	string	`"user"`, `"builtin"`, or `"rethrow"`
`typecode`	int	see Call Type Constants for possible values

System exceptions always throw at least 2 values, populating the `"err"` and `"desc"` keys of the exception hash, giving the exception string code and the exception description string, respectively, and occassionally, depending on the function, the `"arg"` key may be populated with supporting information. User exceptions have no restrictions, any values given in the throw statement will be mapped to exception keys as per the table above.

See the on_exit, on_success statement, and on_error statement for statements that allow for exception-safe and exception-dependent cleanup in Qore code.

Classes that assist in exception-safe lock handling are the AutoLock class, the AutoGate class, the AutoReadLock class, and the AutoWriteLock class.

Chapter 31

Signal Handling

Qore implements safe signal handling on UNIX platforms (not available on native Microsoft Windows ports). Signals do not interrupt Qore threads, rather Qore uses a special signal handling thread with TID 0, dedicated to handling signals. The signal handling thread uses very few resources; it stays blocked (using no processor time and very little memory) until a signal with a Qore signal handler is raised; it then executes the handler and resumes waiting for signals.

Because Qore's signal handling thread is not a normal thread, it does not affect num_threads() and does not appear in the list returned by thread_list().

Internally, Qore masks (blocks) all signals in every thread except the signal handling thread. In the signal handling thread, all signals are unmasked, except those with Qore-language handlers, then an internal call to `sigwait()(3)` is made to receive and process signals raised one at a time.

Qore-language signal handlers are installed by passing a signal constant and a closure or call reference to the code to execute when the signal is raised to the set_signal_handler() function. Signal handlers are removed by passing a signal constant to the remove_signal_handler() function.

When a signal has been raised and the signal handler is called, the signal number is passed as the sole argument to the signal handler code.

Signal Handling Functions

Function Name	Description
set_signal_handler()	Sets up a Qore signal handler using a signal number and a call reference.
remove_signal_handler()	Removes a Qore signal handler using a signal number.

See Signal Constants for a list of signal constants and Qore::NameToSignal and Qore::SignalToName for two hash constants that can be used to map signal names to numbers and vice-versa. Note that signal constants are system-dependent; not all signals will be available in all systems; in case of doubt, see your system documentation for information on which signals are available.

The above functions are atomic, meaning that when they return to the caller, the signal handling thread has already acknowledged the changes.

It is not possible to set signal masks per thread; all signals are delivered to the signal handling thread. Signals not handled with a Qore signal handler are handled with their default action. It is not possible to catch `SIGPIPE`. `SIGPIPE` is always ignored in Qore.

Some issues to be aware of in signal handlers:

- Thread-local storage is not persistent in signal handlers; it is deleted after every signal handler is run.
- A signal handler that does not terminate will block the execution of further signal handlers and will block signal handling changes (such as updating the signal mask), resulting in a Qore process that must be killed manually. Because all Qore signal handling code is executed serially in a single thread, Qore signal handlers should execute and return quickly to give time to execute other handlers.

- Signal handlers may install or remove signal handlers using set_signal_handler() or remove_signal_handler(), however in this case, changes to signal handling are made after the signal handler returns.
- Signal handlers cannot call fork(); any attempt to call fork() in a signal handler will result in an an exception.
- fork() (called externally to a signal handler) is handled as follows: the signal handling thread is terminated, fork() is executed, all signals are masked in the primary thread in the new process, then the signal handling thread is resumed in the parent process only. The signal handler thread cannot be reliably started in the child process because pthread_create() is not async-signal safe, therefore signal handling is disabled in the child process. No signals can be received or handled while the signal handling thread is terminated.
- Unhandled exceptions in signal handlers will simply be displayed on stderr as an unhandled exception and will have no other effect on Qore or Qore code (in particular, unhandled exceptions will not cause the signal handling thread to terminate).
- If a signal handler executes the thread_exit statement, execution of the signal handler will terminate immediately, but the signal handling thread will not be stopped. Execution of further signal handlers (including that for the same signal being handled when thread_exit is executed) will not be affected.

Bug it seems that SIGWINCH and SIGINFO cannot be handled on Darwin in Qore's dedicated signal-handling thread; the signals are never delivered to Qore's signal handling thread for some reason despite setting the internal signal masks appropriately. These signals can be handled normally in the main thread on Darwin (in other programs using traditional non-threading signal APIs), but do not work with Qore (on Darwin only when using the pthread_sigmask() and a dedicated signal-handling thread), possibly due to a bug related to signal handling and threading on Darwin.

Chapter 32

I/O Event Handling

Qore supports a simple event-handling mechanism to provide notification and details of socket and network events in higher-level classes. Classes currently supporting events are the Socket, HTTPClient, FtpClient, and File classes.

See Event Constants for a list of all event constants; details about each event are documented in the following sections.

Event information is placed on the event queue (which must be a Queue object) in the form of a hash. Each event has at least the following keys:

Event Hash Common Keys

Key	Value
`event`	This key holds the event code; see information for individual events in the following sections
`source`	This key holds the event source code
`id`	The value of this key is a unique integer that can be used to uniquely identify the object generating the event.

32.1 EVENT_PACKET_READ

Event

Qore::EVENT_PACKET_READ

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after a network packet is received.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_PACKET_READ
`source`	Qore::SOURCE_SOCKET, indicating the Socket class
`id`	A unique integer ID for the underlying socket object
`read`	The number of bytes read in the packet.

total_read	The total number of bytes read in the read loop.
[total_to_read]	The total number of bytes to read in the read loop (this key is only present if the total number of bytes to read is known).

32.2 EVENT_PACKET_SENT

Event

Qore::EVENT_PACKET_SENT

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after a network packet is sent.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_PACKET_SENT
source	Qore::SOURCE_SOCKET, indicating the Socket class
id	A unique integer ID for the underlying socket object
socket	The file descriptor number of the socket.
sent	The number of bytes sent in the packet.
total_sent	The total number of bytes sent in the send loop.
total_to_send	The total number of bytes to send in the send loop.

32.3 EVENT_HTTP_CONTENT_LENGTH

Event

Qore::EVENT_HTTP_CONTENT_LENGTH

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised immediately after an HTTP header is received containing a content length header line, but before the message body is received.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HTTP_CONTENT_LENGTH
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
id	A unique integer ID for the underlying socket object
len	The number of bytes given for the content length.

32.4 EVENT_HTTP_CHUNKED_START

Event

Qore::EVENT_HTTP_CHUNKED_START

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised after receiving an HTTP header with the `Transfer-Encoding` header set to `chunked` and before the chunked data is read.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_HTTP_CHUNKED_START
`source`	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
`id`	A unique integer ID for the socket object.

32.5 EVENT_HTTP_CHUNKED_END

Event

Qore::EVENT_HTTP_CHUNKED_END

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised after all chunked data is read from the socket.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_HTTP_CHUNKED_END
`source`	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
`id`	A unique integer ID for the socket object.

32.6 EVENT_HTTP_REDIRECT

Event

Qore::EVENT_HTTP_REDIRECT

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised after a redirect response is received from an HTTP server.

The event hash contains the following keys:

Key	Value

event	Qore::EVENT_HTTP_REDIRECT
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
id	A unique integer ID for the socket object.
location	The redirect location given by the HTTP server
[status_message]	Any status message sent by the HTTP server; if no message was sent, then this key will not be present in the event hash.

32.7 EVENT_CHANNEL_CLOSED

Event

Qore::EVENT_CHANNEL_CLOSED

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after the socket is closed.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_CHANNEL_CLOSED
source	Qore::SOURCE_SOCKET, indicating the Socket class
id	A unique integer ID for the underlying socket object

32.8 EVENT_DELETED

Event

Qore::EVENT_DELETED

Source

Qore::SOURCE_SOCKET

Description

This event is raised when the socket object is deleted.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_DELETED
source	Qore::SOURCE_SOCKET, indicating the Socket class
id	A unique integer ID for the underlying socket object

32.9 EVENT_FTP_SEND_MESSAGE

Event

Qore::EVENT_FTP_SEND_MESSAGE

Source

Qore::SOURCE_FTPCLIENT

Description

This event is raised immediately before a message is sent on the FTP control channel.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_FTP_SEND_MESSAGE
`source`	Qore::SOURCE_FTPCLIENT, indicating the FtpClient class
`id`	A unique integer ID for the underlying socket object
`command`	A string giving the FTP command sent (ex: `"RETR"`).
[arg]	The argument to the command; if no argument is sent, then this key will not be present.

32.10 EVENT_FTP_MESSAGE_RECEIVED

Event

Qore::EVENT_FTP_MESSAGE_RECEIVED

Source

Qore::SOURCE_FTPCLIENT

Description

This event is raised immediately after a message is received on the FTP control channel.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_FTP_MESSAGE_RECEIVED
`source`	Qore::SOURCE_FTPCLIENT, indicating the FtpClient class
`id`	A unique integer ID for the underlying socket object
`command`	A string giving the FTP command sent (ex: `"RETR"`).
[arg]	The argument to the command; if no argument is sent, then this key will not be present.

32.11 EVENT_HOSTNAME_LOOKUP

Event

Qore::EVENT_HOSTNAME_LOOKUP

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately before a hostname lookup is made.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HOSTNAME_LOOKUP
source	Qore::SOURCE_SOCKET, indicating the Socket class
id	A unique integer ID for the underlying socket object
name	A string giving the name to be looked up.

32.12 EVENT_HOSTNAME_RESOLVED

Event

Qore::EVENT_HOSTNAME_RESOLVED

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after a successful hostname resolution.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HOSTNAME_RESOLVED
source	Qore::SOURCE_SOCKET, indicating the Socket class
id	A unique integer ID for the underlying socket object
type	The type of address for the socket; one of the Network Address Family Constants
typename	A descriptive name for the address family (ex: "ipv4", "ipv6")
address	A string giving the network address the name was resolved to.

32.13 EVENT_HTTP_SEND_MESSAGE

Event

Qore::EVENT_HTTP_SEND_MESSAGE

Source

Qore::SOURCE_HTTPCLIENT or Qore::SOURCE_SOCKET

Description

This event is raised immediately before an HTTP message is sent.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HTTP_SEND_MESSAGE
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class, or Qore::SOURCE_SOCKET, indicating the Socket class

message	The first string in the HTTP message (ex: GET / HTTP/1.1).
headers	a hash of all headers to send in the message.

32.14 EVENT_HTTP_MESSAGE_RECEIVED

Event

Qore::EVENT_HTTP_MESSAGE_RECEIVED

Source

Qore::SOURCE_HTTPCLIENT or Qore::SOURCE_SOCKET

Description

This event is raised immediately after an HTTP message is received.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HTTP_MESSAGE_RECEIVED
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class, or Qore::SOURCE_SOCKET, indicating the Socket class
headers	A hash of all headers received in the message, plus the following headers giving additional information about the message: - "http_version": giving the HTTP protocol version in the message - "status_code": giving the HTTP status code if the message is a response - "status_message": giving any HTTP status message if the message is a response - "method": giving the HTTP method if the message is a request - "path": providing the path in request messages.

32.15 EVENT_HTTP_FOOTERS_RECEIVED

Event

Qore::EVENT_HTTP_FOOTERS_RECEIVED

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised immediately after HTTP footers are received after receiving chunked data.

The event hash contains the following keys:

Key	Value

event	Qore::EVENT_HTTP_FOOTERS_RECEIVED
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
headers	A hash of all footers received in the message.

32.16 EVENT_HTTP_CHUNKED_DATA_RECEIVED

Event

Qore::EVENT_HTTP_CHUNKED_DATA_RECEIVED

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised immediately after chunked data is received.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HTTP_CHUNKED_DATA_RECEIVED
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
read	An integer giving the number of bytes read in the chunk.
total_read	An integer giving the total number of bytes of chunked data read in the current message.

32.17 EVENT_HTTP_CHUNK_SIZE

Event

Qore::EVENT_HTTP_CHUNK_SIZE

Source

Qore::SOURCE_HTTPCLIENT

Description

This event is raised immediately after chunk information is received providing the size of the next chunk.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_HTTP_CHUNK_SIZE
source	Qore::SOURCE_HTTPCLIENT, indicating the HTTPClient class
size	An integer giving the number of bytes in the next chunk.
total_read	An integer giving the total number of bytes of chunked data read in the current message.

32.18 EVENT_CONNECTING

Event

Qore::EVENT_CONNECTING

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately before a socket connection is attempted.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_CONNECTING
`source`	Qore::SOURCE_SOCKET, indicating the Socket class
`id`	A unique integer ID for the underlying socket object
`type`	The type of address for the socket; one of the Network Address Family Constants
`typename`	A descriptive name for the address family (ex: `"ipv4"`, `"ipv6"`)
`address`	A string giving the target address (ex: `"::1"`, `"192.168.20.4"`)
`target`	The target address for the connection.
[port]	The target port for the connection; if not applicable for the address family then this hash key is not included.

32.19 EVENT_CONNECTED

Event

Qore::EVENT_CONNECTED

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after a socket connection is established.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_CONNECTED
`source`	Qore::SOURCE_SOCKET, indicating the Socket class
`id`	A unique integer ID for the underlying socket object

32.20 EVENT_START_SSL

Event

Qore::EVENT_START_SSL

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately before SSL negotiation is attempted.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_START_SSL
`source`	Qore::SOURCE_SOCKET, indicating the Socket class
`id`	A unique integer ID for the underlying socket object

32.21 EVENT_SSL_ESTABLISHED

Event

Qore::EVENT_SSL_ESTABLISHED

Source

Qore::SOURCE_SOCKET

Description

This event is raised immediately after SSL negotiation has been successfully established.

The event hash contains the following keys:

Key	Value
`event`	Qore::EVENT_SSL_ESTABLISHED
`source`	Qore::SOURCE_SOCKET, indicating the Socket class
`id`	A unique integer ID for the underlying socket object
`cipher`	A string giving the name of the cipher algorithm used for the connection.
`cipher_version`	A string giving the version of the cipher algorithm used for the connection.

32.22 EVENT_OPEN_FILE

Event

Qore::EVENT_OPEN_FILE

Source

Qore::SOURCE_FILE

Description

This event is raised immediately before a file is opened.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_OPEN_FILE
source	Qore::SOURCE_FILE, indicating the File class
id	A unique integer ID for the File object
filename	The file's name.
flags	The flags used to open the file.
mode	The mode to open the file with.
encoding	The character encoding given used for reading from or writing to the file.

32.23 EVENT_FILE_OPENED

Event

Qore::EVENT_FILE_OPENED

Source

Qore::SOURCE_FILE

Description

This event is raised immediately after a file has been successfully opened.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_FILE_OPENED
source	Qore::SOURCE_FILE, indicating the File class
id	A unique integer ID for the File object
filename	The file's name.
flags	The flags used to open the file.
mode	The mode to open the file with.
encoding	The character encoding given used for reading from or writing to the file.

32.24 EVENT_DATA_READ

Event

Qore::EVENT_DATA_READ

Source

Qore::SOURCE_FILE

Description

This event is raised immediately after data is read from a file.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_DATA_READ
source	Qore::SOURCE_FILE, indicating the File class

id	A unique integer ID for the File object
read	The number of bytes read from the file.
total_read	The total number of bytes read in the read loop.
total_to_read	The total number of bytes to read in the read loop.

32.25 EVENT_DATA_WRITTEN

Event

Qore::EVENT_DATA_WRITTEN

Source

Qore::SOURCE_FILE

Description

This event is raised immediately after data is written from a file.

The event hash contains the following keys:

Key	Value
event	Qore::EVENT_DATA_WRITTEN
source	Qore::SOURCE_FILE, indicating the File class
id	A unique integer ID for the File object
written	The number of bytes written to the file.
total_written	The total number of bytes written in the write loop.
total_to_write	The total number of bytes to write in the write loop.

Chapter 33

qore Executable Command-Line Processing

This section describes command-line processing for the `qore` binary; see Qore::GetOpt for the class allowing for command-line processing in Qore code

Qore will scan the command-line for the options in the following table. Arguments after the script name will be passed to the script in the global `$ARGV` variable as a list of options. `$ARGV[0]` will be the first option and will not be the script name. If no script name is given and the `-exec` option is not used, then Qore code is read from standard input. The `$QORE_ARGV` variable will have the original Qore command-line in it, however

Usage: `qore [`*options*`] [`*script_file_name*`]`

Parse Option Command-Line Parameters

Long Param	Short	Description
`--define=`*arg*	`-D`	Creates and optionally sets a value for a parse define
`--load=`*arg*	`-l`	Loads a module immediately. The argument can be a module/feature name or an absolute path to the module
`--no-global-vars`	`-G`	Disallows the use of global variables. Equivalent to parse option Qore::PO_NO_GLOBAL_VARS and the %no-global-vars directive
`--no-subroutine-defs`	`-S`	Disallows subroutine (function) definitions. Equivalent to parse option Qore::PO_NO_SUBROU↩TINE_DEFS and the %no-subroutine-defs directive
`--no-thread-control`	`-R`	Disallows any thread control operations (background operator and thread_exit statement, for example). Equivalent to parse option Qore::PO_NO_THREAD↩_CONTROL and the %no-thread-control directive

`--no-thread-classes`	n/a	Disallows access to thread classes (for example, the Mutex class, Gate class, Queue class, etc). Equivalent to parse option Qore::PO_NO_THREAD_CLASSES and the %no-thread-classes directive
`--no-threads`	`-T`	Disallows access to both thread control and thread classes (equivalent to the `-no-thread-control` and `-no-thread-classes` options documented above). Equivalent to parse option Qore::PO_NO_THREADS and the %no-threads directive
`--no-top-level`	`-L`	Disallows top level code. Equivalent to parse option Qore::PO_NO_TOP_LEVEL_STATEMENTS and the %no-top-level directive
`--no-class-defs`	n/a	Disallows class definitions. Equivalent to Qore::PO_NO_CLASS_DEFS and the %no-class-defs directive
`--no-namespace-defs`	n/a	Disallows new namespace definitions. Equivalent to Qore::PO_NO_NAMESPACE_DEFS and the %no-namespace-defs directive
`--no-external-process`	`-E`	Disallows any access to external processes (with system(), backquote(), exec(), the backquote operator, etc). Equivalent to parse option Qore::PO_NO_EXTERNAL_PROCESS and the %no-external-process directive
`--no-process-control`	`-P`	Disallows access to functions that would affect the current process (exit(), fork(), etc). Equivalent to parse option Qore::PO_NO_PROCESS_CONTROL and the %no-process-control directive
`--no-filesystem`	`-F`	Disallows access to the local filesystem; puts the %no-filesystem parse option in effect. Equivalent to parse option code Qore::PO_NO_FILESYSTEM and the %no-filesystem parse directive

`--no-constant-defs`	n/a	Disallows constant definitions. Equivalent to parse option Qore::PO_NO_CONSTANT_DEFS and the %no-constant-defs directive
`--no-network`	`-Y`	Disallows access to the network; puts the %no-network parse option in effect. Equivalent to parse option code Qore::PO_NO_NETWORK and the %no-network parse directive
`--no-new`	`-N`	Disallows use of the new operator. Equivalent to parse option Qore::PO_NO_NEW and the %no-new directive
`--no-database`	`-D`	Disallows use of database functionality. Equivalent to parse option Qore::PO_NO_DATABASE and the %no-database directive
`--no-child-restrictions`	`-I`	Allows child program objects to have parse option restrictions that are not a strict subset of the parents. Equivalent to parse option Qore::PO_NO_CHILD_PO_RESTRICTIONS and the %no-child-restrictions directive
`--require-our`	`-O`	Requires global variables to be declared with our prior to use (similar to perl's `use strict vars` pragma). Equivalent to parse option Qore::PO_REQUIRE_OUR and the %require-our directive
`--lock-options`	`-K`	Prohibits further changes to parse options (equivalent to the %lock-options directive)
`--lock-warnings`	`-A`	Prohibits further changes to the warning mask. Equivalent to parse option Qore::PO_LOCK_WARNINGS and the %lock-warnings directive
`--enable-all-warnings`	`-W`	Enables all warnings. Equivalent to the %enable-all-warnings directive
`--enable-warning=`*arg*	`-w`	Enables the named warning. Equivalent to the %enable-warning directive
`--list-warnings`	`-i`	Lists all valid warnings in Qore and exits immediately

Miscellaneous Command-Line Parameters

Long Param	Short	Description
`--exec=`*arg*	`-e`	parses and executes the argument text as a Qore program. If this option is specified then any script given on the command-line will be ignored

`--exec-class[=`*arg*`]`	`-x`	instantiates the class with the same name as the program (with the directory path and extension stripped); also turns on –no-top-level. If the program is read from `stdin` or from the command line, an argument must be given specifying the class name
`--show-module-errors`	`-m`	Shows any errors loading Qore modules
`--charset=`*arg*	`-c`	Sets the default character encoding for the program
`--show-charset=`*arg*	`-s`	Shows a list of all known character encodings
`--show-aliases`	`-a`	Shows a list of all known character encoding aliases
`--help`	`-h`	Shows help text
`--version`	`-v`	Shows Qore library version information and exits

There are two additional options available with debugging versions of Qore as follows:

Description of Debugging Command-Line Parameters

Long Param	Short	Description
`--debug=`*arg*	`-d`	Turns on Qore debugging output (output to `stderr`). Higher arg numbers give more output. This option is only available with DEBUG builds
`--trace`	`-t`	Turns on Qore tracing (output to `stderr`). This option is only available with DEBUG builds

Chapter 34

Parse Directives

Qore supports the use of parse directives in order to set parsing options, load modules, control warnings, and include other files.

Parse directives that set parsing options can be used any time parse options have not been locked on a Program object. They are used most often when it's not possible or desirable to set the parse options in the qore command line.

Parse Directives

Directive	Description
%allow-bare-refs	Prohibits the use of the "$" character in variable names, method calls, and object member references. This makes Qore scripts appear superficially more like C++ or Java programs. This parse option is set by default with %new-style. Name resolution is made in the following order when this option is set: local variables, class constants and static class vars (when parsing in class code), global variables, and (non-class) constants. Note that implicit arguments are still referenced with the "$" character even when this parse option is set; see also %require-dollar Since Qore 0.8.1

%append-include-path	Appends the given directories to qore's include path; also performs environment variable substitution
%append-module-path	Appends the given directories to qore's module path; also performs environment variable substitution Since Qore 0.8.6
%assume-global	Resets the default Qore behavior of assuming global variable scope when variables are first referenced if no my or our is present; use after %assume-local to reset the default parsing behavior. This parse option is also set with %old-style Since Qore 0.8.4
%assume-local	Assume local variable scope when variables are first referenced if no my or our is present. When used with %allow-bare-refs, local variables without my must be declared with a data type restriction (can be any). This parse option is set by default with %new-style; see also %assume-global Since Qore 0.8.1
%define	Creates and optionally sets a value for a parse define Since Qore 0.8.3
%disable-all-warnings	Turns off all warnings
%disable-warning *warning-code*	Disables the named warning until %enable-warning is encountered with the same code or %enable-all-warnings is encountered
%else	Allows for parsing an alternate block when used with the %ifdef or %ifndef parse directives (for conditional parsing based on parse defines) Since Qore 0.8.3
%enable-all-warnings	Turns on all warnings
%enable-warning *warning-code*	Enables the named warning
%endif	Closes a conditionally-parsed block started by the %ifdef or %ifndef parse directives Since Qore 0.8.3
%endtry	Closes a %try-module block Since Qore 0.8.6
%exec-class *class_name*	Instantiates the named class as the application class; also turns on %no-top-level. If the program is read from stdin or from the command line, an argument must be given specifying the class name
%ifdef	Opens a conditionally-parsed block; if the given parse define is defined, then the block after the %ifdef is parsed until either an %else or an %endif Since Qore 0.8.3

%ifndef	Opens a conditionally-parsed block; if the given parse define is not defined, then the block after the %ifdef is parsed until either an %else or an %endif Since Qore 0.8.3
%include *file_name*	Starts parsing *file_name* immediately. Parsing resumes with the current input after *file_name* has been completely parsed
%lockdown	Made up of %no-external-access, %no-threads, and %no-io; equivalent to parse option Qore::PO_LOCKDOWN and the `-lockdown` command line option
%lock-options	Prohibits further changes to parse options (equivalent to the `--lock-options` command line option)
%lock-warnings	Prohibits further changes to the warning mask (equivalent to the `--lock-warnings` command line option)
%new-style	Sets both %allow-bare-refs and %assume-local. These two options together make programming in Qore superficially more like programming in C++ or Java programs; use this if you dislike programming with the `"$"` sign, for example; see also %old-style Since Qore 0.8.1
%old-style	Resets default Qore parsing behavior by setting %require-dollar and %assume-global; this option is the opposite of %new-style Since Qore 0.8.4
%no-class-defs	Disallows class definitions; equivalent to Qore::PO_NO_CLASS_DEFS and the `--no-class-defs` command line option
%no-child-restrictions	Allows child program objects to have parse option restrictions that are not a strict subset of the parents'; equivalent to parse option Qore::PO_NO_CHILD_PO_RESTRICTIONS and the `--no-child-restrictions` command line option
%no-constant-defs	Disallows constant definitions; equivalent to parse option Qore::PO_NO_CONSTANT_DEFS and the `--no-constant-defs` command line option
%no-database	Disallows access to database functionality (for example the Datasource class; equivalent to parse option Qore::PO_NO_DATABASE and the `--no-database` command line option
%no-external-access	made up of %no-process-control, %no-network, %no-filesystem, %no-database, %no-external-info, and %no-modules; equivalent to parse option Qore::PO_NO_EXTERNAL_ACCESS and the `--no-external-access` command line option

%no-external-info	Disallows any access to functionality that provides external information (see %no-external-info for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_EXTERNAL_INFO and the `--no-external-info` command line option
%no-external-process	Disallows any access to external processes (see %no-external-process for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_EXTERNAL_PROCESS and the `--no-external-process` command line option
%no-filesystem	Disallows access to the local filesystem; equivalent to parse option Qore::PO_NO_FILESYSTEM and the `--no-filesystem` command line option
%no-global-vars	Disallows the use of global variables; equivalent to parse option Qore::PO_NO_GLOBAL_VARS and the `--no-global-vars` command line option
%no-gui	Disallows functionality that draws graphics to the display; equivalent to parse option Qore::PO_NO_GUI and the `-pno-gui` command line option
%no-io	Made up of %no-gui | %no-terminal-io | %no-filesystem | %no-network | %no-database; equivalent to parse option Qore::PO_NO_IO and the `-no-io` command line option
%no-locale-control	Disallows access to functionality that changes locale information (see no-locale-control for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_LOCALE_CONTROL and the `--no-locale-control` command line option
%no-modules	Disallows loading modules with the %requires or %try-module parse directive or at runtime with load_module(); equivalent to Qore::PO_NO_MODULES and the `-pno-modules` command line option Since Qore 0.8.4
%no-namespace-defs	Disallows new namespace definitions; equivalent to Qore::PO_NO_NAMESPACE_DEFS and the `--no-namespace-defs` command line option
%no-network	Disallows access to the network; equivalent to parse option Qore::PO_NO_NETWORK and the `--no-network` command line option
%no-new	Disallows use of the new operator; equivalent to parse option Qore::PO_NO_NEW and the `--no-new` command line option
%no-process-control	Disallows access to functions that would affect the current process (see %no-process-control for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_PROCESS_CONTROL and the `--no-process-control` command line option

%no-subroutine-defs	Disallows subroutine (function) definitions; equivalent to parse option Qore::PO_NO_SUBROUTINE_DEFS and the `--no-subroutine-defs` command line option
%no-terminal-io	Disallows access to terminal I/O (see %no-terminal-io for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_TERMINAL_IO and the `-pno-terminal-io` command line option
%no-thread-classes	Disallows access to thread classes (see %no-thread-classes for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_THREAD_CLASSES and the `--no-thread-classes` command line option
%no-thread-control	Disallows access to thread control operations (see %no-thread-control for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_THREAD_CONTROL and the `--no-thread-control` command line option
%no-thread-info	Disallows any access to functionality that provides threading information (see %no-thread-info for a list of features not available with this parse option); equivalent to parse option Qore::PO_NO_THREAD_INFO and the `--no-thread-info` command line option
%no-threads	Disallows access to all thread control operations and thread classes (equivalent to `--no-thread-control` and `--no-thread-classes` together); equivalent to parse option Qore::PO_NO_THREADS and the `--no-threads` command line option
%no-top-level	Disallows top level code; equivalent to parse option Qore::PO_NO_TOP_LEVEL_STATEMENTS and the `--no-top-level` command line option
%perl-bool-eval	Set to mimic perl's boolean evaluation; this is the default with qore >= 0.8.6; prior to this version, by default qore used strict mathematic boolean evaluation, where any value converted to 0 is False and otherwise it's is True. As of Qore 0.8.6+, this parse option is only needed if %strict-bool-eval is set. When this option is set, boolean evaluation follows the same rules as Qore::zzz8valuezzz9::val(); see also %strict-bool-eval Since Qore 0.8.6

%push-parse-options	Stores parse options so that they will be restored when the current file is done parsing; use in include files to ensure parse options are set appropriately for the file being parsed
%require-dollar	Resets the default Qore behavior where the use of the "$" character in variable names, method calls, and object member references is required; use after %allow-bare-refs to reset the default parsing behavior Since Qore 0.8.4
%require-our	Requires global variables to be declared with our prior to use (like perl's `use strict vars` pragma); equivalent to parse option Qore::PO_REQUIRE_OUR and the `--require-our` command line option
%require-prototypes	Requires type declarations for all function and method parameters and return types. Variables and object members do not need to have type declarations; equivalent to parse option Qore::PO_REQUIRE_PROTOTYPES and the `--require-prototypes` command line option Since Qore 0.8.0
%require-types	Requires type declarations for all function and method parameters, return types, variables, and object members; equivalent to parse option Qore::PO_REQUIRE_TYPES and the `--require-types` command line option; also implies %strict-args Since Qore 0.8.0
%requires *feature* [<|<=|=|>=|> *version*]	If the named feature is not already present in Qore, then the `QORE_MODULE_DIR` environment variable is used to provide a list of directories to seach for a module with the same name (*feature*`[-api-`*x*`.`*y*`].qmod` for binary modules or *feature*`.qm` for user modules). If the module is not found, then the qore default module directory is checked. This directive must be used to load modules providing parse support (i.e. modules providing classes, constants, functions, etc that are resolved at parse time). If version information is provided, then it is compared with the module's version information, and if it does not match a parse exception is raised. See also Qore::load_module() for a function providing run-time module loading and %try-module for a similar parse directive that allows module loading errors to be handled at runtime

%set-time-zone	Sets the time zone for the current program from a UTC offset (with format "+/-[00[:00[:00]]"; ":" characters are optional) or a time zone region name (ex: "Europe/Prague") Since Qore 0.8.3
%strict-args	Prohibits access to builtin functions and methods flagged with RUNTIME_NOOP and also causes errors to be raised if excess arguments are given to functions that do not access excess arguments Since Qore 0.8.0
strict-bool-eval	Sets qore's default strict mathematic boolean evaluation mode, where any value converted to 0 is False and otherwise it's is True; this was how qore behaved by default prior to v0.8.6. Equivalent to parse option Qore::PO_STRICT_BOOLEAN_EVAL and the -pstrict-bool-eval command line option. See also %perl-bool-eval Since Qore 0.8.6
%try-module (var_decl) *feature* [<|<=|=|>=|> *version*]	If the named feature is not already present in Qore, then the QORE_MODULE_DIR environment variable is used to provide a list of directories to seach for a module with the same name (*feature*[-api-*x.y*].qmod for binary modules or *feature*.qm for user modules). If the module is not found, then the qore default module directory is checked. If an error occurs loading the module, then the variable declared after %try-module is instantiated with the exception information, and the code up to the %endtry parse declaration is parsed into the program, allowing for the qore script/program to handle module loading errors at parse time for modules that must be loaded at parse time. If version information is provided, then it is compared with the module's version information, and if it does not match a parse exception is raised that is handled like any other load error. See also Qore::load_module() for a function providing run-time module loading and %requires for a similar declaration that does not allow for parse-time module loading error handling. Since Qore 0.8.6

34.1 %allow-bare-refs

Parse Directive:

```
%allow-bare-refs
```

Command Line:

`--allow-bare-refs, -B`

Parse Option Constant:

Qore::PO_ALLOW_BARE_REFS

Description:

Prohibits the use of the `"$"` character in variable names, method calls, and object member references. This makes Qore scripts appear superficially more like C++ or Java programs. This parse option is set by default with %new-style and is the opposite of %require-dollar.

Name resolution is made in the following order when this option is set:

- local variables
- class constants (when parsing in class code)
- static class variables (when parsing in class code)
- global variables
- (non-class) constants

Note that implicit arguments are still referenced with the `"$"` character even when this parse option is set.

Since

Qore 0.8.1

See also

%require-dollar

34.2 %append-include-path

Parse Directive:

`%append-include-path` *dir1[:dir2...]*

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Appends the given directories to qore's include path; also performs environment variable substitution. Only directories that exist and are readable are added to the include path.

See also

%append-module-path

34.3 %append-module-path

Parse Directive:

`%append-module-path` *dir1[:dir2...]*

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Appends the given directories to qore's module path; also performs environment variable substitution. Only directories that exist and are readable are added to the module path.

Since

Qore 0.8.6

See also

%append-include-path

34.4 %assume-global

Parse Directive:

`%assume-global`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Resets the default Qore behavior of assuming global variable scope when variables are first referenced if no my or our is present; use after %assume-local to reset the default parsing behavior. This parse option is also set with %old-style

Since

Qore 0.8.4

See also

%assume-local

34.5 %assume-local

Parse Directive:

```
%assume-local
```

Command Line:

```
--assume-local
```

Parse Option Constant:

Qore::PO_ASSUME_LOCAL

Description:

Assume local variable scope when variables are first referenced if no my or our is present. When used with %allow-bare-refs, local variables without my must be declared with a data type restriction (can be any). This parse option is set by default with %new-style and is the opposite of %assume-global

Since

Qore 0.8.1

See also

%assume-global

34.6 %define

Parse Directive:

```
%define
```

Command Line:

```
--define, -D
```

Parse Option Constant:

n/a

Description:

Creates and optionally sets a value for a parse define

Since

Qore 0.8.3

34.7 %disable-all-warnings

Parse Directive:

```
%disable-all-warnings
```

Command Line:

n/a, warnings are disabled by default

Parse Option Constant:

n/a

Description:

Disables all warnings while parsing. See Warnings for more information.

34.8 %disable-warning

Parse Directive:

```
%disable-warning
```

Command Line:

n/a, warnings are disabled by default

Parse Option Constant:

n/a

Description:

Disables the named warning while parsing. See Warnings for more information.

34.9 %else

Parse Directive:

```
%else
```

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Allows for parsing an alternate block when used with the %ifdef or %ifndef parse directives (for conditional parsing based on parse defines)

Since

Qore 0.8.3

34.10 %enable-all-warnings

Parse Directive:

```
%enable-all-warnings
```

Command Line:

```
--enable-all-warnings, -W
```

Parse Option Constant:

n/a

Description

Enables all warnings while parsing. See Warnings for more information.

34.11 %enable-warning

Parse Directive:

```
%enable-warning
```

Command Line:

```
--enable-warning, -w
```

Parse Option Constant:

n/a

Description:

Enables the named warning while parsing. See Warnings for more information.

34.12 %endif

Parse Directive:

```
%endif
```

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Closes a conditionally-parsed block started by the %ifdef or %ifndef parse directives (for conditional parsing based on parse defines)

Since

Qore 0.8.3

34.13 %endtry

Parse Directive:

```
%endtry
```

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Closes a %try-module block

Since

Qore 0.8.6

34.14 %exec-class

Parse Directive:

`%exec-class`

Command Line:

`--exec-class, -x`

Parse Option Constant:

n/a

Description:

Executes the named class as the application class and turns on %no-top-level as well.

34.15 %ifdef

Parse Directive:

`%ifdef`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Opens a conditionally-parsed block; if the given parse define is defined, then the block after the `%ifdef` is parsed until either an %else or an %endif

Since

Qore 0.8.3

34.16 %ifndef

Parse Directive:

`%ifndef`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Opens a conditionally-parsed block; if the given parse define is not defined, then the block after the `%ifndef` is parsed until either an %else or an %endif

Since

Qore 0.8.3

34.17 %include

Parse Directive:

`%include`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Includes a file to be parsed. If the path is not absolute (i.e. starting with `""/"`), then files are searched first in the directory of the currently-executing script (if known), then in each path in the environment variable `QORE_INCLUDE_DIR`.
If the first character of the include target is `'$'`, then environment variable substitution is performed on the leading environment variable

Since

Qore 0.8.9 added environment variable substitution

34.18 %lockdown

Parse Directive:

`%lockdown`

Command Line:

`--lockdown`

Parse Option Constant:

Qore::PO_LOCKDOWN

Description:

The most restrictive parse restriction, making any external access, threading, i/o, etc illegal. Made up of %no-external-access, %no-threads, and %no-io

34.19 %lock-options

Parse Directive:

`%lock-options`

Command Line:

`--lock-options, -K`

Parse Option Constant:

n/a

Description:

Prohibits further changes to parse options.

34.20 %lock-warnings

Parse Directive:

`%lock-warnings`

Command Line:

`--lock-warnings, -A`

Parse Option Constant:

Qore::PO_LOCK_WARNINGS

Description:

Prohibits further changes to the warning mask.

34.21 %new-style

Parse Directive:

`%new-style`

Command Line:

`--new-style, -n`

Parse Option Constant:

Qore::PO_NEW_STYLE

Description:

Sets both %allow-bare-refs and %assume-local. These two options together make programming in Qore superficially more like programming in C++ or Java programs; use this if you dislike programming with the "$" sign, for example.

See also

%old-style

Since

Qore 0.8.1

34.22 %old-style

Parse Directive:

`%old-style`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Resets default Qore parsing behavior by setting %require-dollar and %assume-global; this option has the opposite effect of %new-style

See also

%new-style

Since

Qore 0.8.4

34.23 %no-child-restrictions

Parse Directive:

`%no-child-restrictions`

Command Line:

`--no-child-restrictions, -I`

Parse Option Constant:

Qore::PO_NO_CHILD_PO_RESTRICTIONS

Description:

Allows child program objects to have parse option restrictions that are not a strict subset of the parents'.

34.24 %no-class-defs

Parse Directive:

`%no-class-defs`

Command Line:

`--no-class-defs`

Parse Option Constant:

Qore::PO_NO_CLASS_DEFS

Description:

Disallows new class definitions. Any use of the reserved word `class` will result in a parse exception.

34.25 %no-constant-defs

Parse Directive:

```
%no-constant-defs
```

Command Line:

```
--no-constant-defs
```

Parse Option Constant:

Qore::PO_NO_CONSTANT_DEFS

Description:

Disallows new constant definitions. Any use of the reserved word `const` will result in a parse exception.

34.26 %no-database

Parse Directive:

```
%no-database
```

Command Line:

```
--no-database
```

Parse Option Constant:

Qore::PO_NO_DATABASE

Description:

Disallows access to database functionality. Currently this means that access to the following classes is restricted:

- Qore::SQL::Datasource
- Qore::SQL::DatasourcePool
- Qore::SQL::SQLStatement

34.27 %no-external-access

Parse Directive:

```
%no-external-access
```

Command Line:

```
--no-external-access
```

Parse Option Constant:

Qore::PO_NO_EXTERNAL_ACCESS

Description:

This option is made up of %no-process-control, %no-network, %no-filesystem, %no-database, %no-external-info, and %no-modules

See also

%lockdown

34.28 %no-external-info

Parse Directive:

```
%no-external-info
```

Command Line:

```
--no-external-info
```

Parse Option Constant:

Qore::PO_NO_EXTERNAL_INFO

Description:

Disallows any access to functionality that provides external information. The following features are unavailable with this option:

- getaddrinfo()
- getegid()
- getenv()
- geteuid()
- getgid()
- getgrgid()
- getgrgid2()
- getgrnam()
- getgrnam2()
- getgroups()
- gethostbyaddr()
- gethostbyaddr_long()
- gethostbyname()
- gethostbyname_long()
- gethostname()
- getpid()
- getppid()
- getpwnam()
- getpwnam2()
- getpwuid()
- getpwuid2()
- getuid()
- unsetenv()

34.29 %no-external-process

Parse Directive:

```
%no-external-process
```

Command Line:

```
--no-external-process, -E
```

Parse Option Constant:

Qore::PO_NO_EXTERNAL_PROCESS

Description:

Disallows any access to external processes. The following features are unavailable with this option:

- backquote operator
- system()
- kill()
- exec()
- backquote()

34.30 %no-filesystem

Parse Directive:

```
%no-filesystem
```

Command Line:

```
--no-filesystem, -F
```

Parse Option Constant:

Qore::PO_NO_FILESYSTEM

Description:

Disallows any access to the external filesystem. The following features are unavailable with this option:

- Qore::Dir
- Qore::File
- Qore::File::hstat(string)
- Qore::File::hlstat(string)
- Qore::File::lstat(string)
- Qore::File::stat(string)
- Qore::File::statvfs(string)
- Qore::FtpClient::get()
- Qore::FtpClient::put()
- chdir()
- chmod()
- chown()
- getcwd()
- getcwd2()
- glob()
- hlstat()
- hstat()
- is_bdev()
- is_cdev()
- is_dev()
- is_dir()

- is_executable()
- is_file()
- is_link()
- is_pipe()
- is_readable()
- is_socket()
- is_writable()
- is_writeable()
- lchown()
- lstat()
- mkdir()
- mkfifo()
- readlink()
- rename()
- rmdir()
- stat()
- statvfs()
- unlink()

34.31 %no-global-vars

Parse Directive:

```
%no-global-vars
```

Command Line:

```
--no-global-vars, -G
```

Parse Option Constant:

Qore::PO_NO_GLOBAL_VARS

Description:

Disallows the use of global variables.

34.32 %no-gui

Parse Directive:

```
%no-gui
```

Command Line:

```
--no-gui, -set-parse-option=no-gui, -pno-gui
```

Parse Option Constant:

Qore::PO_NO_GUI

Description:

Disallows the use of functionality that draws graphics to the display (this functionality is not implemented in the qore library; only implemented in modules).

34.33 %no-io

Parse Directive:

```
%no-io
```

Command Line:

```
--no-io, -set-parse-option=no-io, -pno-io
```

Parse Option Constant:

Qore::PO_NO_IO

Description:

Made up of %no-gui | %no-terminal-io | %no-filesystem | %no-network | %no-database

34.34 %no-locale-control

Parse Directive:

```
%no-locale-control
```

Command Line:

```
--no-locale-control, -P
```

Parse Option Constant:

Qore::PO_NO_LOCALE_CONTROL

Description:

Disallows access to functions that would affect the locale settings of the current program. The following features are unavailable with this option:

- Qore::TimeZone::set()
- Qore::TimeZone::setUTCOffset()
- Qore::TimeZone::setRegion()

34.35 %no-modules

Parse Directive:

```
%no-modules
```

Command Line:

```
-pno-modules
```

Parse Option Constant:

Qore::PO_NO_MODULES

Description:

Disallows loading modules with the %requires or %try-module directives or at runtime with load_module()

Since

Qore 0.8.4

34.36 %no-namespace-defs

Parse Directive:

```
%no-namespace-defs
```

Command Line:

```
--no-namespace-defs, -M
```

Parse Option Constant:

Qore::PO_NO_NAMESPACE_DEFS

Description:

Disallows new namespace definitions.

34.37 %no-network

Parse Directive:

```
%no-network
```

Command Line:

```
--no-network, -Y
```

Parse Option Constant:

Qore::PO_NO_NETWORK

Description:

Disallows any access to the network. The following features are unavailable with this option:

- Qore::FtpClient
- Qore::HTTPClient
- Qore::Socket

34.38 %no-new

Parse Directive:

```
%no-new
```

Command Line:

```
--no-new, -N
```

Parse Option Constant:

Qore::PO_NO_NEW

Description:

Disallows use of the new operator.

34.39 %no-process-control

Parse Directive:

```
%no-process-control
```

Command Line:

```
--no-process-control, -P
```

Parse Option Constant:

Qore::PO_NO_PROCESS_CONTROL

Description:

Disallows access to functions that would affect the current process. The following features are unavailable with this option:

- abort()
- chdir()
- exit()
- fork()
- load_module()
- remove_signal_handler()
- setegid()
- setenv()
- seteuid()
- setgid()
- setgroups()
- setsid()
- set_signal_handler()
- setuid()
- sleep()
- srand()
- umask()
- unsetenv()
- usleep()

34.40 %no-subroutine-defs

Parse Directive:

```
%no-subroutine-defs
```

Command Line:

```
--no-subroutine-defs, -S
```

Parse Option Constant:

Qore::PO_NO_SUBROUTINE_DEFS

Description:

Disallows subroutine (function) definitions.

34.41 %no-terminal-io

Parse Directive:

```
%no-terminal-io
```

Command Line:

```
--no-terminal-io,--set-parse-option=no-terminal-io, -pno-terminal-io
```

Parse Option Constant:

Qore::PO_NO_TERMINAL_IO

Description:

Disallows access to terminal input and output. The following features are unavailable with this option:

- Qore::TermIOS
- Qore::TermIOS::getWindowSize()
- Qore::stdin
- Qore::stdout
- Qore::stderr
- f_printf()
- f_vprintf()
- flush()
- print()
- printf()
- vprintf()

34.42 %no-thread-classes

Parse Directive:

```
%no-thread-classes
```

Command Line:

```
--no-thread-classes
```

Parse Option Constant:

Qore::PO_NO_THREAD_CLASSES

Description:

Disallows access to thread classes. The following features are unavailable with this option:

- Qore::Thread::AbstractSmartLock class
- Qore::Thread::AutoGate class
- Qore::Thread::AutoLock class
- Qore::Thread::AutoReadLock class
- Qore::Thread::AutoWriteLock class
- Qore::Thread::Condition class
- Qore::Thread::Counter class
- Qore::Thread::Gate class
- Qore::Thread::Mutex class
- Qore::Thread::Queue class
- Qore::Thread::RWLock class
- Qore::Thread::ThreadPool class

34.43 %no-thread-control

Parse Directive:

```
%no-thread-control
```

Command Line:

```
--no-thread-control, -R
```

Parse Option Constant:

Qore::PO_NO_THREAD_CONTROL

Description:

Disallows access to thread control operations. The following features are unavailable with this option:

- background operator
- thread_exit statement
- delete_all_thread_data()
- delete_thread_data()
- getAllThreadCallStacks()
- get_all_thread_data()
- get_thread_data()
- mark_thread_resources()
- remove_thread_data()
- save_thread_data()
- throwThreadResourceExceptions()
- throw_thread_resource_exceptions_to_mark()

34.44 %no-thread-info

Parse Directive:

```
%no-thread-info
```

Command Line:

```
--no-thread-info
```

Parse Option Constant:

Qore::PO_NO_THREAD_INFO

Description:

Disallows access to functionality that provides information about threading. The following features are unavailable with this option:

- get_all_thread_data() function
- get_thread_data() function
- gettid() function
- getAllThreadCallStacks() function
- num_threads() function
- thread_list() function

34.45 %no-threads

Parse Directive:

`%no-threads`

Command Line:

`--no-threads, -T`

Parse Option Constant:

Qore::PO_NO_THREADS

Description:

Disallows access to all thread control operations and thread classes (equivalent to %no-thread-control and %no-thread-classes together).

34.46 %no-top-level

Parse Directive:

`%no-top-level`

Command Line:

`--no-top-level, -L`

Parse Option Constant:

Qore::PO_NO_TOP_LEVEL_STATEMENTS

Description:

Disallows top level code.

34.47 %perl-bool-eval

Parse Directive:

`%perl-bool-eval`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Set to mimic perl's (and Python's) boolean evaluation; this is now the default (as of qore version 0.8.6); prior to this version, by default qore used strict mathematic boolean evaluation, where any value converted to 0 is False and otherwise it's is True.

As of Qore 0.8.6+, `%perl-bool-eval` is the default behavior and therefore only needs to be set if %strict-bool-eval was set first.

When this option is set, boolean evaluation follows the same rules as Qore::zzz8valuezzz9::val(); for example:

```
my string $str = "hello";
if ($str)
    printf("string is: %y\n", $str);
```

With this option set (the default), the above prints out:

```
string is: "hello"
```

When it's not set (i.e. %strict-bool-eval is set), nothing is printed out as the string expression evaluates to False due to qore's old strict mathematical boolean evaluation.

Basically with this option set, qore's boolean evaluation becomes like perl's and Python's, whereas any expression that has the following values is False: NOTHING, string `"0"` and empty strings, integer, float, and number `0` (zero), absolute date `1970-01-01Z` (ie the start of the epoch with an offset of 0), relative date `PT0H` (ie any relative date with a 0 duration), NULL, empty binary objects, empty hashes, and empty lists. All other values are True.

Note

also affects the boolean(any) function

See also

%strict-bool-eval

Since

Qore 0.8.6

34.48 %push-parse-options

Parse Directive:

`%push-parse-options`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Stores parse options so that they will be restored when the current file is done parsing; use in include files to ensure parse options are set appropriately for the file being parsed.

For example:

```
%push-parse-options
%new-style
```

34.49 %require-dollar

Parse Directive:

`%require-dollar`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Resets the default Qore behavior where the use of the `"$"` character in variable names, method calls, and object member references is required; use after %allow-bare-refs to reset the default parsing behavior

Since

Qore 0.8.4

34.50 %require-our

Parse Directive:

```
%require-our
```

Command Line:

```
--require-our, -O
```

Parse Option Constant:

Qore::PO_REQUIRE_OUR

Description:

Requires global variables to be declared with our prior to use (recommended to use for all larger scripts/programs).

34.51 %require-prototypes

Parse Directive:

```
%require-prototypes
```

Command Line:

```
--require-prototypes
```

Parse Option Constant:

Qore::PO_REQUIRE_PROTOTYPES

Description:

Requires type declarations for all function and method parameters and return types. Variables and object members do not need to have type declarations with this option.

See also

%require-types, which is a superset of this option.

34.52 %require-types

Parse Directive:

`%require-types`

Command Line:

`--require-types`

Parse Option Constant:

Qore::PO_REQUIRE_TYPES

Description:

Requires type declarations for all function and method parameters, return types, variables, and object members.

See also

%require-prototypes, which is a subset of this option.

34.53 %requires

Parse Directive:

`%requires` *feature* `[<|<=|=|>=|>` *version*`]`

Command Line:

`--load, -l`

Parse Option Constant:

n/a

Description:

Loads a Qore module immediately. The parse directive can be used to load a module during parsing, and the command line version can be used to load a module before parsing.

From Qore 0.7.1, you can specify a comparison operator (one of `<, <=, =, >=, or >`) and version information after the module name as well. Version numbers are compared via integer comparisons of each element, where elements are separated by a `"."`. If one of the versions does not have as many elements as another, the missing elements are assumed to be `"0"` (i.e. version `"1.0"` compared with version `"1.0.1"` will be extended to `"1.0.0"`).

For example:

```
%requires oracle >= 1.0.1
```

This will load the oracle module if it is at least version 1.0.1.

Note that there is one special feature name: `"qore"`. This pseudo-feature can be used to check the minimum Qore version; if this feature is requested with version information, then the Qore library's version information is used for the version number comparison.

See also

%try-module for a similar parse directive that allows module loading errors to be caught and handled at parse-time

34.54 %set-time-zone

Parse Directive:

```
%set-time-zone
```

Command Line:

```
--time-zone, -z
```

Parse Option Constant:

none; this is set by calling Qore::TimeZone::setRegion() or Qore::TimeZone::setUTCOffset()

Description:

Sets the time zone for the current program from a UTC offset (with format "+/-00[:00[:00]]"; ":"" characters are optional) or a time zone region name (ex: \c "Europe/Prague").

34.55 %strict-args

Parse Directive:

```
%strict-args
```

Command Line:

```
--strict-args
```

Parse Option Constant:

Qore::PO_STRICT_ARGS

Description:

Prohibits access to builtin functions and methods flagged with RUNTIME_NOOP and also causes errors to be raised if excess arguments are given to functions that do not access excess arguments.

34.56 %strict-bool-eval

Parse Directive:

```
%strict-bool-eval
```

Command Line:

```
-pstrict-bool-eval
```

Parse Option Constant:

Qore::PO_STRICT_BOOLEAN_EVAL

Description:

When set, this option enables qore's default strict mathematic boolean evaluation, where any value converted to 0 is False and otherwise it's is True.

This option takes effect at runtime.

For example:

```
my string $str = "hello";
if ($str)
    printf("string is: %y\n", $str);
```

If `%strict-bool-eval` is set, then Qore's boolean evaluation will evaluate `"hello"` in a strict mathematical sense and convert it to 0, which is False, so nothing would be printed out, but when `%perl-bool-eval` is enabled, this will be True (following the rules for Qore::zzz8stringzzz9::val()).

Qore's strict mathematical boolean evaluation was the default prior to Qore 0.8.6, however it was considered non-intuitive and a design bug, and therefore the default behavior of the language was changed to a more intuitive form of boolean evaluation as documented in %perl-bool-eval.

Note

also affects the boolean(any) function

See also

perl-bool-eval

Since

Qore 0.8.6

34.57 %try-module

Parse Directive:

`%try-module` [(var_decl)] *feature* `[<|<=|=|>=|>` *version*`]`

Command Line:

n/a

Parse Option Constant:

n/a

Description:

Loads a Qore module immediately; if an error occurs loading the module, then the variable declared in parentheses after `%try-module` is instantiated with the exception information, and the code up to the %endtry parse declaration is parsed into the program, allowing for the qore script/program to handle module loading errors at parse time for modules that must be loaded at parse time.

A comparison operator (one of `<`, `<=`, `=`, `>=`, `or` `>`) and version information can be optionally given after the module name as well. Version numbers are compared via integer comparisons of each element, where elements are separated by a `"."`. If one of the versions does not have as many elements as another, the missing elements are assumed to be `"0"` (i.e. version `"1.0"` compared with version `"1.0.1"` will be extended to `"1.0.0"`). Version errors are handled in the same was as any other module loading error.

For example:

```
%try-module($ex) some_module > 2.0
    printf("error loading module %y: %s: %s\n", $ex.arg, $ex.err, $ex.desc);
    exit(1);
%endtry
```

This will load the "some_module" module if it is at least version 2.0, and print an error message and exit gracefully if it cannot be loaded.

The variable declaration in parentheses is optional; use the variant without the exception variable in code where parse option PO_NO_TOP_LEVEL_STATEMENTS is set, for example. In this case, for example, other parse options can be set before the %endtry directive.

Here is an example without a variable declaration in parentheses:

```
%try-module some_module > 2.0
%define DontHaveSomeModule
%endtry
```

Note that there is one special feature name: "qore". This pseudo-feature can be used to check the minimum Qore version; if this feature is requested with version information, then the Qore library's version information is used for the version number comparison.

See also

%requires for a similar parse directive where errors are handled with the default parse exception handler

Since

Qore 0.8.6; the variant without the variable declaration was introduced in Qore 0.8.6.1

Chapter 35

Warnings

Warnings give the programmer information about possible errors in Qore code.

Warnings can be enabled using the `-W` command-line option (see qore Executable Command-Line Processing for more information) or by using the %enable-all-warnings or %enable-warning parse directives.

Command Line Example

```
prompt$ qore -wdeprecated -wduplicate-hash-key script.q
```

Qore Code Example

```
%enable-warning deprecated
%enable-warning duplicate-hash-key
%disable-warning excess-args
```

List of All Warnings

- call-with-type-errors
- deprecated
- duplicate-block-vars
- duplicate-global-vars
- duplicate-hash-key
- duplicate-local-vars
- excess-args
- invalid-operation
- module-only
- non-existent-method-call
- return-value-ignored
- undeclared-var
- unknown-warning
- unreachable-code
- unreferenced-variable
- warning-mask-unchanged

35.1 call-with-type-errors

Raised when the parser determines that the argument types of a function or method call are such that the operation is guaranteed to produce a constant value

Since

Qore 0.8.0

35.2 deprecated

Raised when deprecated functionality is accessed.

See also

"Deprecated" Functions

Since

Qore 0.8.0

35.3 duplicate-block-vars

Raised when a program declares a local variable more than once in the same block; note that this is not a warning but rather an error when %assume-local or %new-style parse options are set

Since

Qore 0.8.2

35.4 duplicate-global-vars

Raised when a program declares a global variable more than once

Since

Qore 0.5.2

35.5 duplicate-hash-key

Raised when an immediate hash is declared and at least one of the keys is repeated

Since

Qore 0.8.0

35.6 duplicate-local-vars

This warning is raised when a local variable with the same name is declared in a subblock (ie another local variable with the same name is reachable in the same lexical scope); note that this warning can raise false positives if the programmer is used to redeclaring the same variable names in subblocks

See also

duplicate-block-vars

Since

Qore 0.5.2

35.7 excess-args

Raised when a function or method call is made with more arguments than are used by the function or method

Since

Qore 0.8.0

35.8 invalid-operation

Raised when the parser determins that the types of an operation are such that the operation is guaranteed to produce no value; this warning can only be raised when type information is available at parse time

Since

Qore 0.8.0

35.9 module-only

This warning is raised when a feature that is only valid in a user module is used in code that is not in a user module, for example, declaring a class or namespace The "public" Keyword

Since

Qore 0.8.4

35.10 non-existent-method-call

Warning is raised when the given method cannot be found in the class at parse time; this is a warning because the object could be a subclass that has the given method implemented, in which case the call will succeed at run time. Use the cast<>() operator to avoid this warning

Since

Qore 0.8.0

35.11 return-value-ignored

Raised when a function or method call is made with no side effects and the return value is ignored

Since

Qore 0.8.0

35.12 undeclared-var

This warning is raised when a program uses a variable that has not been declared with my or our

Since

Qore 0.5.2

35.13 unknown-warning

This warning is raised when a program tries to enable or disable an unknown warning

Since

Qore 0.5.2

35.14 unreachable-code

Raised when code is defined that can never be executed (for example, code following a return or thread_exit statement)

Since

Qore 0.5.2

35.15 unreferenced-variable

Raised when a variable is declared but never referenecd

Since

Qore 0.8.2

35.16 warning-mask-unchanged

This warning is raised when a program tries to change the warning mask with parse options, but the warnings are locked

Since

Qore 0.5.2

Chapter 36

Keywords

Keyword Types

Type	Description
Soft 1	Keywords of this type cannot be used as static variable names or constant names but can be used as function or method names if they are followed immediately by an open parenthesis (with no whitespace between the keyword and the open parenthesis), and they can be referenced as unquoted object or hash member names only if they are immediately preceded by the . operator (again with no whitespace between the "." and the keyword):
Soft 2	Keywords of this type cannot be used as static variable names or constant names but can be used as function or method names if they are followed by an open parenthesis (whitespace is allowed between the keyword and the open parenthesis), and they can be referenced as unquoted object or hash member names if they are immediately preceded by the . operator (again with no whitespace between the "." and the keyword):
Soft 3	Keywords of this type be referenced as unquoted object or hash member names if preceded immediately by the "." operator, however they may not be used as function or class method names or static variable names or constant names

Keywords

Keywords	Type	Description
background	Soft 1	for the background operator
case	Soft 1	used in switch statements
chomp	Soft 1	for the chomp operator
class	Soft 1	used when declaring classes
delete	Soft 1	for the delete operator
exists	Soft 1	for the exists operator
final	Soft 2	for final method declarations or final class declarations
foldl	Soft 1	for the foldl operator

foldr	Soft 1	for the foldr operator
map	Soft 1	for the map operator
pop	Soft 1	for the pop operator
private	Soft 1	class members, class methods, and static class variables can all be declared private to the class with this keyword, additionally classes can be privately inherited using this keyword as well
public	Soft 1	class members, class methods, and static class variables can all be declared public to the class with this keyword, additionally this keyword can be used when declaring a publically inherited class as well
push	Soft 1	for the push operator
new	Soft 1	for the new operator
select	Soft 1	for the select operator
shift	Soft 1	for the shift operator
splice	Soft 1	for the splice operator
trim	Soft 1	for the trim operator
unshift	Soft 1	for the unshift operator
default	Soft 2	used in switch statements
static	Soft 2	used when declaring static methods and static variables in a class
break	Soft 3	for the break statement
catch	Soft 3	used with try catch statements
const	Soft 3	used to declare constants and class constants
continue	Soft 3	for the continue statement
do	Soft 3	for the do statement
else	Soft 3	used with the if statement
for	Soft 3	for the for statement (not to be confused with the foreach list iterator statement)
foreach	Soft 3	for the foreach list iterator statement (not to be confused with the for statement)
if	Soft 3	for the if statement
in	Soft 3	used with the foreach list iterator statement and the find expression
inherits	Soft 3	used when declaring class inheritance
instanceof	Soft 3	for the instanceof operator
namespace	Soft 3	used when declaring namespaces
my	Soft 3	used when declaring local variables

elements	Soft 3	for the elements operator
find	Soft 3	for the find expression
keys	Soft 3	for the keys operator
returns	Soft 3	used when using the deprecated syntax for declaring method or function return types
abstract	Soft 3	This keyword is reserved for future implementation
deprecated	Soft 3	when declaring functions or methods as deprecated
context	Soft 3	for the context statement
summarize	Soft 3	for the summarize statement
subcontext	Soft 3	for the subcontext statement
sortBy	Soft 3	used with context, summarize, and subcontext statements
sortDescendingBy	Soft 3	used with context, summarize, and subcontext statements
by	Soft 3	used with summarize statements
switch	Soft 3	for the switch statement
return	Soft 3	for the return statement
rethrow	Soft 3	for the rethrow statement
on_error	Soft 3	for the on_error statement
on_exit	Soft 3	for the on_exit statement
on_success	Soft 3	for the on_success statement
our	Soft 3	used when declaring global variables
sub	Soft 3	used when declaring functions
synchronized	Soft 3	used when declaring synchronized functions or methods
thread_exit	Soft 3	for the thread_exit statement
throw	Soft 3	for the throw statements
try	Soft 3	for the try statements
where	Soft 3	used with context, summarize, and subcontext statements
while	Soft 3	for the while statement and used with the do statement

Chapter 37

Release Notes

37.1 Qore 0.8.11.1

Release Summary

Minor release to meet packaging standards of Fedora and OpenSUSE.

37.1.1 New Features in Qore

- new functions:
  - Qore::getgroups()
  - Qore::setgroups()
- module directory handling changed
  - user modules are now stored in $prefix/share/qore-modules/$version
  - $prefix/share/qore-modules is also added to the module path
  - version-specific module directories are added first, then the "generic" directories

37.2 Qore 0.8.11.1

Release Summary

Minor bugfix release for UNIX, major bugfixes for Windows

37.2.1 New Features in Qore

- added the Qore::AFMap and Qore::AFStrMap constants
- `WebUtil` updates:
  - added logic to the default file serving code to determine if the file is a binary or text file from the MIME type
- HTTPClient::sendWithSendCallback() and HTTPClient::sendWithSendCallback() updated such that if a response is received while the chunked send operation is still in progress, an error is assumed, the send operation is aborted, and the response header is read immediately

37.2.2 Bug Fixes in Qore

- Windows fixes:
  - fixed TimeZone copying to use the standard name instead of the display name so that the info can be found in the registry
  - `Util` module fixes:
    - fixed get_random_string() on Windows
    - fixed absolute_path_windows()
  - `HttpServer` module fixes:
    - when binding a wildcard address with AF_UNSPEC on Windows with HttpServer::addListeners() and both IPv6 and IPv4 addresses are returned, bind both addresses since Windows doesn't direct the IPv4 requests to the wildcard-bound IPv6 listener
  - fixed file reading by always opening in binary mode
  - added support for the WSAECONNABORTED socket error
  - replaced Mime::MultiPartMessage::getRandomString() with Util::get_random_string() to make it work on Windows
- fixed a bug in the DBI layer where calling SQLStatement::describe() would crash when called with an older module that did not implement this method
- other fixes in the `Util` module (in addition to the Windows-specific fixes above):
  - fixed parse_to_qore_value() with hashes with a comma in the first key name
  - read from /dev/urandom instead of /dev/random since reads from the latter can block for long periods to fill the entropy pool
- do not start signal thread after a fork() if signal handling is enabled, pthread_create() is not async-signal safe (on FreeBSD at least this reliably causes segfaults)

37.3 Qore 0.8.10

Release Summary

Major release with many many bugfixes and new features such as much improved HTTP and REST support (ex: chunked transfer support + new client and server classes for REST support for chunked transfers and data streaming), improved DB support (ex: new `Schema` module, `SqlUtil` improvements), and much more.

37.3.1 New Features in Qore

- better HTTP support; support for chunked sends and receives for streaming data over HTTP and other improvements:
  - HTTPClient::sendWithSendCallback()
  - HTTPClient::sendWithRecvCallback()
  - HTTPClient::sendWithCallbacks()
  - HTTPClient::setPersistent()
  - Socket::readHTTPChunkedBodyBinaryWithCallback()
  - Socket::readHTTPChunkedBodyWithCallback()
  - Socket::sendHTTPMessageWithCallback()
  - Socket::sendHTTPResponseWithCallback()
  - Socket::pendingHttpChunkedBody()
- added a minimum body size threshold for compression to `HttpServer`

- `RestClient` module updates:
  - configurable content encoding for send request message bodies is now supported (ie optional compression)
- new user modules:
  - `Schema`: for DB-independent schema management
- new public C++ socket performance instrumentation API
- new functions:
  - Qore::close_all_fd()
- new constants:
  - ESRCH: search error
- getModuleHash() and getModuleList() no longer return the `"filename"` key when run in a Program context with Qore::PO_NO_EXTERNAL_INFO set
- `SqlUtil` updates:
  - added insert operator support; for example, for inserting with values from sequences
  - added new upsert constant maps
  - added static SqlUtil::AbstractSqlUtilBase::getDatasourceDesc() method
  - added new Table::insertFromSelect*() variants taking Table arguments
  - added SqlUtil::Table::checkExistence() method
  - added support for the `"forupdate"` select option
- `OracleSqlUtil` updates:
  - fixed selects with "limit" but no "offset"
  - convert date/time values to timestamps with microseconds resolution instead of dates with second resolution when dynamically inserting values as strings in SQL (binding by value not affected)
- `CsvUtil` module updates:
  - added the `"write-headers"` option to CsvUtil::AbstractCsvWriter and subclasses to enable headers to be suppressed
  - added the `"optimal-quotes"` option to CsvUtil::AbstractCsvWriter and subclasses to enable more efficient csv output (now the default)
- added AbstractDatasource::currentThreadInTransaction() which is reimplemented as Datasource::currentThreadInTransaction() and DatasourcePool::currentThreadInTransaction(); the base class method throws an exception when called; it was not added as an abstract method in order to not break existing subclasses of AbstractDatasource
- enhanced module license support
  - module license strings may now be specified in binary and user modules
  - Qore::getModuleHash() and Qore::getModuleList() now report license information for each module

37.3.2 Bug Fixes in Qore

- fixed an issue with class constant parse initialization where invalid recursive class constant definition parse exceptions could be raised and in some cases also crashes could result
- `SmtpClient` module: fixed missing username and missing password errors
- fixed a bug where a qore switch statement with no case conditions and only a default label would erroneously never have it's default code executed
- fixed a reference leak related to exception handling with invalid arguments with Qore::Socket::setWarningQueue() and Qore::HTTPClient::setWarningQueue()
- fixed several bugs where the parse location could be reported incorrectly for type errors regarding in-object variable references
- fixed a bug where an error could result with Condition::wait() with timeouts $>$ 2147483648ms
- fixed bugs handling "bigint" and "double precision" column types with schema alignments with the `PgsqlSqlUtil` module
- fixed a bug handling parse initialization of constant values requiring run-time evaluation after other parse exceptions have been raised that could cause a parse-time crash
- fixed a bug where qore could crash on exit with certain openssl versions by calling ERR_remove_state(0) in the main thread's cleanup function
- fixed a bug where qore could crash on exit due to user module destruction not taking into consideration user module dependencies
- fixed a bug in schema management in `SqlUtil` where excessively verbose column aliases were used that caused errors when automatically updating columns with existing rows and new default values and non-null contraints with PostgreSQL databases
- fixed a bug where a call reference to an abstract object method returned from an abstract class could be executed even though it must have been instantiated by a concrete subclass
- fixed a bug where a valid call reference to a private object method was created within the class, then in some cases an object protection exception was raised when the call reference was called outside the class
- fixed a bug in the `RestClient` module when the yaml binary module is not available
- fixed programmatic select queries with "limit" but no "offset" in `OracleSqlUtil`
- fixed a bug in Qore::Program::importFunction() where only the committed function list was checked when importing functions with a specific target namespace path
- fixed a bug in Qore::Program::importClass() where only the committed class list was checked when importing functions with a specific target namespace path
- fixed a bug when parsing subnamespaces into a parent namespace where the subnamespace already exists (either in the committed list or in the pending list)
- fixed a memory and reference leak caused by recursive references when closures encapsulating an object's scope are assigned to or accessible from members of the object by making references to the object from within a closure encapsulating the object's state weak references instead of strong references
- fixed schema information classes when the "string-numbers" driver option is enabled
- fixed crashing bugs in Qore::get_thread_data() in certain use cases
- fixed a bug in `SqlUtil` where select and row iterator operations could fail with certain select hash arguments without a `"columns"` entry but where column names were otherwise required
- fixed a bug in HTTP response parsing where case-signficant comparisons where being made with certain critical header values
- fixed a bug handling thread cancellation with the ThreadPool class
- fixed several race conditions and potential deadlocks in ThreadPool destruction with active threads in the pool

37.4 Qore 0.8.9

Release Summary

Major release with many new features and also many bugfixes.

37.4.1 New Features in Qore

- `CsvUtil` module updates:
  - new classes:
    - `CsvAbstractIterator`: base abstract iterator class for iterating line-based CSV data
    - `CsvDataIterator`: iterator class allowing for CSV string data to be processed line by line on a record basis
    - `AbstractCsvWriter`: a base class for new CSV writer implementations
    - `CsvFileWriter`: CSV file writer class
    - `CsvStringWriter`: CSV in memory writer class
  - implemented support for allowing subclasses of CsvFileIterator to implement support for other custom types
  - no need to set `"headers"` in the constructor if `"fields"` are set; headers are assumed to be the field labels in the same order
- added the Qore::encode_url() function with `RFC 3986 section 2.1` compliance
- Qore::decode_url() function updated to decode UTF-8 encoded characters according to `RFC 3986 section 2.1`
- added get_byte_size() and get_marketing_byte_size() to the `Util` module
- the error message now includes the module path used for the search when a module cannot be found in the module path
- DatasourcePool enhancements:
  - new method: DatasourcePool::clearWarningCallback()
  - new method: DatasourcePool::setWarningCallback()
  - new method: DatasourcePool::getUsageInfo()
  - new method: DatasourcePool::setErrorTimeout()
  - new method: DatasourcePool::getErrorTimeout()
  - new method: DatasourcePool::setEventQueue()
  - new method: DatasourcePool::clearEventQueue()
  - the new methods allow for monitoring DatasourcePool objects for pool contention issues (in case the pool needs to be resized), for throwing an exception if a connection is not acquired within the error timeout period (new default: 2 minutes), and for DBI drivers to raise warnings on an event queue that can be monitored in a separate thread
  - additionally connection acquisition statistics are tracked and returned in DatasourcePool::getUsageInfo() (total requests, hits, maximum wait time)
- Datasource enhancements:
  - new method: Datasource::setEventQueue()
  - new method: Datasource::clearEventQueue()
  - the new methods allow for DBI drivers to raise warnings on an event queue that can be monitored in a separate thread
- Socket enhancements:

  - new method: Socket::setWarningQueue()
  - new method: Socket::clearWarningQueue()
  - new method: Socket::getUsageInfo()
  - new method: Socket::clearStats()

- FtpClient enhancements:
  - new method: FtpClient::setWarningQueue()
  - new method: FtpClient::clearWarningQueue()
  - new method: FtpClient::getUsageInfo()
  - new method: FtpClient::clearStats()

- `SmtpClient` module updates:
  - optimized connection and login code; HELO/EHLO and authorization are performed when connecting only, not before each email
  - added support for socket performance instrumentation and warning events

- `Pop3Client` module updates:
  - added support for socket performance instrumentation and warning events

- `TelnetClient` module updates:
  - added support for socket performance instrumentation and warning events

- `WebSocketClient` module updates:
  - added support for socket performance instrumentation and warning events

- `RestClient` module updates:
  - use the new Qore::encode_url() function to encode URL paths to ensure that valid requests are sent when spaces, percent characters, and non-ascii characters are used in the URL path
  - set the character encoding in the `Content-Type` request header when sending strings
  - set the `Accept` header correctly in requests (previously only indicated yaml (`"text/x-yaml"`) as an acceptible response encoding)

- `RestHandler` module updates:
  - added support for the `OPTIONS` method
  - return a 400 `"Bad Request"` error if an unsupported HTTP method is used in a REST Call

- added new `UpsertInsertOnly` upsert strategy to `SqlUtil`

- new pseudo-methods:
  - Qore::zzz8valuezzz9::sizep(): returns True if the type can return a non-zero size (True for containers including binary objects and strings, False for everything else)
  - Qore::zzz8stringzzz9::getLine(): finds lines in a string buffer

- `Mime.qm` module updates:
  - added mime type for WSDL files (`"application/wsdl+xml"`)
  - added mappings for `"xls"` and `"xlst"` extensions to MimeTypeXml

- added new modules:
  - `Mapper`: data mapping module
  - `TableMapper`: data mapping module using `SqlUtil` and `Mapper` to map to an SQL table target

- the %include parse directive now supports environment variable substitution at the beginning of the file path

37.4.2 Bug Fixes in Qore

- fixed a crashing bug when HTTP messages with duplicate `Connection`, `Content-Encoding`, `Transfer-Encoding`, `Location`, or `Content-Type` headers is received
- fixed a bug parsing octal character constants in the lexer when octal digits followed the octal constant (ex: `"\0441"` where the `"1"` would cause an error)
- allow escaping "$" character in regular expression substitution target strings, previously it was impossible to output a literal "$" + a digit, since this would be interpreted as a numbered input pattern expression
- fixed a bug in the HTTPClient::getURL() and HTTPClient::getProxyURL() methods where the URL's path was not given with a leading "/" character
- `CsvUtil` module fixes:
  - fixed `"date"` field handling with empty input (now maps to 1970-01-01)
  - fixed CsvDataIterator::next() when header_lines $>$ 0 and working with empty input data
- added support for compiling on OSX Mavericks
- fixed an infinitely recursive call in Table::del() in `SqlUtil`
- fixed a bug in v∗printf() where `'%'` was not handled correctly in all cases
- fixed bugs in microseconds and milliseconds() with large arguments
- fixed a bug where a call to a call reference to a static method across a program boundary with local variables as arguments would cause a crash due to improper setting of the program context before the arguments are evaluated
- fixed a bug in Datasource::copy() method where implicitly-set options were not carried over into the new object
- fixed a bug in the DatasourcePool class where implicitly-opened connections would not be guaranteed to have the same server time zone setting as the initial connections (for example, could cause problems with server timezone settings if running in a program context with a different local time zone attribute)
- fixed bugs in `SqlUtil` generating "create table" and "align table" SQL with DBs where unique indexes automatically create unique constraints (ex: MySQL)
- fixed a bug in lchown() where chown() was used interally instead of lchown()
- fixed a bug in `PgsqlSqlUtil` retrieving sequence values with Database::getNextSequenceValue()
- fixed an off-by-one memory bug in Qore::date(string, string) parsing a 4-digit date mask
- fixed memory leaks in class member and class static variable management
- fixed memory leaks when an entire class has to be rolled back due to parse errors and the class has pending static variables
- fixed memory leaks in constant handling with values containing call references
- fixed a memory leak in constant destruction with parse rollbacks when the constant value was NULL
- fixed an error in the rounding heuristic for arbitrary-precision numeric values that could produce invalid results (ex: 34.9n ∗ 100 = 34902n)
- enforce PO_NO_FILESYSTEM with the %include directive
- fixed a bug managing object private data in complex inheritance cases where the same class may be inherited with virtual private data and also real private data
- fixed a bug in socket timeout handling with select() errors
- fixed a memory leak in handling abstract methods when multiple abstract methods with the same name but different signatures were declared in a class

37.5 Qore 0.8.8

Release Summary

Major new features and bug fixes with a particular focus on enhanced HTTP capabilities and enhanced database processing

37.5.1 Changes That Can Affect Backwards-Compatibility

- Fixed method resolution order; it's now possible to call pseudo-methods directly on classes that implement methodGate() methods
- Added the following abstract methods to AbstractDatasource:
  - AbstractDatasource::getConfigHash()
  - AbstractDatasource::getConfigString()
- "hard" string comparisons now perform encoding conversions if necessary (however as usual different data types cause the comparison to fail)

37.5.2 New Features in Qore

- new user modules delivered with Qore:
  - `RestClient`: Provides a simple API for communicating with HTTP servers implementing `REST` services
  - `RestHandler`: Provides an easy to use interface to the Qore `HttpServer` module for implementing server-side `REST` services
  - `SqlUtil`: Provides a high-level DB-independent API for working with database objects
    - `MysqlSqlUtil`: Provides a high-level DB-independent API for working with MySQL database objects; loaded automatically by the `SqlUtil` module when working with MySQL databases
    - `OracleSqlUtil`: Provides a high-level DB-independent API for working with Oracle database objects; loaded automatically by the `SqlUtil` module when working with Oracle databases
    - `PgsqlSqlUtil`: Provides a high-level DB-independent API for working with PostgreSQL database objects; loaded automatically by the `SqlUtil` module when working with PostgreSQL databases
  - `Util`: Provides a some miscellaneous generally useful routines
  - `WebSocketClient`: Provides a client API for connecting to WebSocket servers
  - `WebSocketHandler`: Provides an interface to the Qore `HttpServer` module for implementing server-side WebSocket services
  - `WebSocketUtil`: Provides common client and server code for implementing WebSocket protocol services in Qore
  - `WebUtil`: Provides server support for implementing complex web services including serving resources with mixed text and Qore code that are automatically rendered on demand
- improvements in existing user modules:
  - much improved `HttpServer` module, better performance, much better RFC compliance, more flexibility
  - new CSV generation class in `CsvUtil`
  - much better message serialization and email attachment handling in the `SmtpClient` and `MailMessage` modules
- there is a new ThreadPool class for implementing thread pools that automatically upscale and downscale within user-defined limits depending on the load placed on them

- it's possible to inherit concrete versions of abstract method variants from a parent class that does not define the abstract method, meaning that concrete variants of an abstract method do not have to be implemented in a direct subclass of the class declaring the abstract method
  this makes using abstract base classes much easier in complex hierarchies using multiple inheritance; now common code can be separated into a single class and inherited by child classes sharing the common implementation
- major Socket read performance increase by implementing internal read buffering (up to 10x faster socket read performance for certain operations, particularly with HTTP methods)
- improved Unicode / UTF-8 support
  - Qore::zzz8stringzzz9::lwr(), Qore::zzz8stringzzz9::upr(), Qore::tolower(string), and Qore::toupper(string) now operate on a very wide range of non-ASCII characters, including Latin, Cyrillic, Greek, Armenian, Georgian, etc characters whereas they were previously limited to working on ASCII characters
  - Qore::zzz8stringzzz9::unaccent() was added which removes accents from strings using a Unicode lookup map from a very wide range of accented Unicode characters to unaccented characters
- new Datasource and DatasourcePool methods:
  - Qore::SQL::Datasource::getConfigHash()
  - Qore::SQL::Datasource::getConfigString()
  - Qore::SQL::DatasourcePool::getConfigHash()
  - Qore::SQL::DatasourcePool::getConfigString()
  - Qore::SQL::DatasourcePool::copy()
- HTTPClient changes:
  - the HTTPClient class is now a subclass of Socket, so all Socket methods can be called on HTTPClient objects, making it easier to implement protocols based on HTTP
  - Qore::HTTPClient::getDefaultPath(): added
  - Qore::HTTPClient::setDefaultPath(): added
  - Qore::HTTPClient::getURL(): changed: now returns NOTHING if no URL is set instead of an invalid URL
- new functions:
  - call_pseudo_args()
  - substr(binary, softint)
  - substr(binary, softint, softint)
- new pseudo methods:
  - Qore::zzz8binaryzzz9::substr(softint)
  - Qore::zzz8binaryzzz9::substr(softint, softint)
  - Qore::zzz8binaryzzz9::toBase64()
  - Qore::zzz8binaryzzz9::toHex()
  - Qore::zzz8binaryzzz9::toString()
  - Qore::zzz8datezzz9::getEpochSeconds()
  - Qore::zzz8datezzz9::getEpochSecondsLocalTime()
  - Qore::zzz8datezzz9::info()
  - Qore::zzz8floatzzz9::abs()
  - Qore::zzz8hashzzz9::compareKeys(hash)
  - Qore::zzz8intzzz9::abs()
  - Qore::zzz8intzzz9::encodeLsb(int)
  - Qore::zzz8intzzz9::encodeMsb(int)

  - Qore::zzz8intzzz9::toUnicode()
  - Qore::zzz8numberzzz9::abs()
  - Qore::zzz8objectzzz9::hasCallableMethod()
  - Qore::zzz8objectzzz9::hasCallableNormalMethod()
  - Qore::zzz8objectzzz9::hasCallableStaticMethod()
  - Qore::zzz8listzzz9::rangeIterator()
  - Qore::zzz8nothingzzz9::rangeIterator()
  - Qore::zzz8stringzzz9::comparePartial()
  - Qore::zzz8stringzzz9::getUnicode()
  - Qore::zzz8stringzzz9::equalPartial()
  - Qore::zzz8stringzzz9::equalPartialPath()
  - Qore::zzz8stringzzz9::toBase64()
  - Qore::zzz8stringzzz9::toHex()
  - Qore::zzz8stringzzz9::unaccent()
  - Qore::zzz8valuezzz9::toNumber()
- other new methods and method changes:
  - added new static methods in the ReadOnlyFile class making it easier to read entire files in one call:
    - Qore::ReadOnlyFile::readTextFile()
    - Qore::ReadOnlyFile::readBinaryFile()
  - changes to catch usage errors with the Counter class:
    - Qore::Thread::Counter::constructor() will throw an exception if called with an argument < 0
    - Qore::Thread::Counter::dec() will now throw an exception if called when the Counter is already at 0
  - Qore::Thread::Queue::empty(): new method
  - Qore::Socket::listen(): now has a new *backlog* parameter; the default backlog queue size was changed from 5 to 20
  - Qore::Socket::getPeerInfo() and Qore::Socket::getSocketInfo(): now takes an optional argument to avoid name lookups
  - Qore::Socket::readHTTPHeaderString(): new method
  - Qore::Dir: all list$*$() methods now take an optional parameter to return a list of file status value hashes plus `"name"` and optionally `"link"` keys for symbolic links; additionally symbolic links are now followed and files and directories are differentiated based on their targets when processing symbolic links
- function changes
  - added optional *start* and *end* parameters to the replace() function
- all data type declarations that optionally accept NOTHING also now accept NULL and map NULL to NOTHING; this makes direct assignments from values derived from SQL queries much easier
- added an optional reference to an integer to the Qore::backquote() function to return the return code of the program executed
- implicit index references now work in the map and select operators with lists and iterators
- the Regular Expression Pattern Extraction Operator now accepts an optional `g` specifier to extract all occurrences of the pattern(s) in a string; also regex_extract() and Qore::zzz8stringzzz9::regexExtract(string, int) now accept Qore::RE_Global to extract all occurrences of the pattern(s) in a string
- the splice and extract operators were extended to work on binary objects as well as lists and strings
- printing out binary values with the `"%y"` format specifier now produces YAML-like output for the binary value
- added path name to error messages in Dir class exception strings

37.5.3 Bug Fixes in Qore

- fixed a bug where the ?: operator could thrown spurious exceptions when parsing because it would return the type of the initial boolean expression as the return type of the operator
- fixed a bug where classes with unimplemented inherited abstract variants would sometimes cause runtime exceptions to be thrown when instantiated but should have instead been caught at parse time
- fixed a parser bug where out-of-line class method definitions could not be defined in a namespace block
- fixed a bug parsing arguments in parse_uri_query() in the `HttpServer` module
- fixed several bugs where parse exceptions could show the wrong source location:
  - with type errors in function calls
  - when resolving global variables
  - in base class constructor arguments
  - for empty blocks with a missing return statement
  - when validating types used with the return statement (also associated warnings)
  - in methods calls
  - in hash value expressions
  - with redeclaring local variable return types
  - in local variable object instantiations
- really fixed the bug thought to be fixed in 0.8.7 "where SSL errors would cause the affected thread to go into a infinite loop using 100% CPU" - this turned out to be easily reproducible on all platforms; when the SSL connection was shut down cleanly by the remote end before a response message was returned, an infinite loop would result
- fixed a bug where it was impossible to output a single '\' character in regex substitution expressions; '\' was taken as an escape character, and '\\' was output literally, now '\\' is output as '\'
- fixed a bug where a parse-time crash would occur when calling the copy() method for a class that does not implement an explicit copy() method
- fixed a bug where arguments passed to a copy method were ignored; now an exception is thrown
- fixed a bug where public members and static variables of privately-inherited classes were incorrectly treated as public attributes of the child class
- fixed a bug where slices could be made of objects from outside the class including private members
- fixed a bug where memberGate() methods were not being respected when taking a slice of an object
- fixed bugs in the integer Socket::recv*() methods where a `SOCKET-CLOSED` exception was not thrown when the remote end closed the connection
- fixed a bug related to out-of-order parse initialization for functions and methods which resulted in the wrong return type being returned for a method with more than 1 variant where the variant could not be matched at parse time
- fixed a bug where a non-variable-reference member of an "our" variable declaration list would cause a crash due to passing the incorrect argument in sprintf()
- fixed sandboxing / protection errors with inherited code; subclasses inheriting code from a parent class with different parse options would cause the child parse options to be used when running the parent class code which caused errors; now parse options are enforced properly on the block level
- fixed the RangeIterator class; it was still abstract due to a missing Qore::RangeIterator::valid() method
- fixed a bug where the wrong error was being returned after a connection reset (remote connection close) in sockets with integer recv*() methods which could in some cases lead to an infinite loop

- fixed a bug where private members of a common base class were not accessible by objects of subclasses sharing the common base class
- fixed many bugs in `CsvUtil` and updated the module version to 1.1
- initialize static openssl crypto locks for multi-threaded openssl library access; without this crashes can result (for example in error queue management)
- fixed a bug where Qore::HTTPClient::getURL() returned an invalid URL when no URL was set; now it returns NOTHING in this case
- fixed a bug managing feature/module lists in inherited Program objects; user modules were listed in the child Program object even though user module code is not imported in child Program objects
- fixed a bug where an invalid guard condition in critical lvalue storage code can cause unreferenced data to be returned while in a lock which can cause a crash in a multithreaded program
- fixed a bug where references were not being written to the output variable if an exception was active when the code block exited
- fixed a bug setting the precision for arbitrary-precision numbers with large exponents (like "1e100n")
- implemented more strict adherence to `RFC 2616` (HTTP 1.1) regarding message-body handling in requests and response message generation and parsing
- fixed a bug with Condition::wait() on Darwin with negative timeout values where a short timeout was used instead of an indefinite wait
- fixed bugs in the `SmtpClient` and `MailMessage` modules where mail messages were being serialized incorrectly if there were no attachments (there was no possibility to set the content transfer encoding) and also where it was not possible to set the content-type for the message body when it was sent as a part of a multipart message
- fixed bugs handling arguments declared as type *reference (reference or nothing)
- fixed bugs in executing code accross Program object barriers with reference arguments
- fixed a bug with the switch statement where character encoding differences would cause strings to mismatch even if they were otherwise identical; now hard comparisons with strings allow for implicit automatic temporary character encoding conversions for the comparison
- fixed a bug where qore failed to set the time zone region correctly when set from /etc/localtime and this file is a relative symlink rather than absolute
- fixed a bug where substr() and Qore::zzz8stringzzz9::substr() were returning NOTHING if the arguments could not be satisifed contrary to the documentation and the declared return type, now an empty string is returned in those cases
- fixed bugs rounding number values between 10 and -10 (non-inclusive) for display, fixed bugs rounding number value regarding digits after the decimal point for display with Qore::NF_Scientific
- fixed a bug in the Qore::Dir class where it was not possible to chdir to the root directory "/"
- fixed a bug where recursive references were allowed and memory leaks would occur due to recursive references; these are now caught at runtime and a `REFERENCE-ERROR` exception is thrown
- fixed a configure bug with bison >= 3
- fixed a bug in the `HttpServer` module when automatically uncompressing supported content-encodings to set the resulting string's character encoding correctly
- fixed a bug in the instanceof operator when working with objects and classes created from different source Program objects
- fixed a bug in *printf() formatting with floating-point and number values where no digits were displayed right of the decimal point unless a specific number of digits was specified in the format string

- fixed the return type of Qore::zzz8boolzzz9::typeCode(); was returning a boolean instead of Qore::NT_BOOLEAN
- fixed a bug there NULL was evaluated as True in a boolean context rather than False
- fixed a bug where Qore::Socket::recvBinary() would ignore the first data read
- fixed starting listeners on UNIX domain sockets on Soalris in the `HttpServer` module
- fixed a bug where number("") was being converted to @NaN@n
- fixed return type of HTTPClient::getConnectionPath()
- fixed several bugs with logical comparison operators and arbitrary-precision numeric values where arbitrary-precision numeric values were not being prioritized as numeric values and also in some cases were being first converted to doubles and then operated on
- fixed a bug in the socket code where the socket close condition was not flagged with SSL connections when writes failed due to the remote end closing the connection; an error would only be raised on the following socket operation
- fixed a mismatched delete/malloc error with time zone initialization and the localtime file

37.6 Qore 0.8.7

Release Summary

Code embedding improvements

37.6.1 Changes That Can Affect Backwards-Compatibility

Fixes for Code Inheritance in Program Objects

The following changes are meant to sanitize code inheritance in child Program objects to fix long-standing design bugs in code encapsulation by addressing the lack of fine-grained control over symbol visibility in inherited code.

- public: The public keyword's usage in modules has now been expanded to provide the same functionality generically in Program objects; if classes, constants, namespaces, functions, or global variables are defined with the public keyword, then these symbols will be inherited into child Program objects as long as no parse options prohibit it.

  This change was made to give programmers complete control over which symbols are inherited in child Program objects, whereas because prior to this change, the control was very course.

- the default behavior of Qore regarding inherting global variables and functions with user variants was changed to be consistent with namespaces, classes, and constants; that is; public symbols are inherited by default.

  The following constants were renamed:

  - `PO_INHERIT_USER_FUNC_VARIANTS` is now: PO_NO_INHERIT_USER_FUNC_VARIANTS
  - `PO_INHERIT_GLOBAL_VARS` is now: PO_NO_INHERIT_GLOBAL_VARS

  This change was made to fix a long-standing design problem with symbol inheritance and make the implementation consistent.

- builtin symbols are no longer inherited from user modules; only independent user symbols; the main change is that if a user module adds new user methods to a builtin class or new user variants to a builtin function, these changes are no longer imported into target Program objects.

File Method Changes

The following methods were updated to throw exceptions on all errors rather than a return code for I/O errors in order to avoid hard to debug conditions due to ignoring I/O errors by forgetting to check the return value on the following methods:

- Qore::File::f_printf()
- Qore::File::f_vprintf()
- Qore::File::print()
- Qore::File::printf()
- Qore::File::vprintf()
- Qore::File::write()
- Qore::File::writei1()
- Qore::File::writei2()
- Qore::File::writei4()
- Qore::File::writei8()
- Qore::File::writei2LSB()
- Qore::File::writei4LSB()
- Qore::File::writei8LSB()

Note that the above changes will hopefully only minimally impact backwards-compatibilty since the change is in error handling, and additionally each of the above methods could also throw an exception if called when the object was not open.

37.6.2 New Features in Qore

- new methods offering code encapsulation enhancements
  - Program::loadModule(): allows modules to be loaded in a Program object directly
  - Program::importClass(): allows classes to be individually imported in Program objects
- new pseudo-methods in Qore::zzz8nothingzzz9 to allow for Qore::zzz8hashzzz9 pseudo-methods to be safely used with NOTHING
  - Qore::zzz8nothingzzz9::firstKey()
  - Qore::zzz8nothingzzz9::firstValue()
  - Qore::zzz8nothingzzz9::hasKey()
  - Qore::zzz8nothingzzz9::hasKeyValue()
  - Qore::zzz8nothingzzz9::keys()
  - Qore::zzz8nothingzzz9::lastKey()
  - Qore::zzz8nothingzzz9::lastValue()
  - Qore::zzz8nothingzzz9::values()
- other new pseudo-methods:
  - Qore::zzz8datezzz9::durationSeconds()
  - Qore::zzz8datezzz9::durationMilliseconds()
  - Qore::zzz8datezzz9::durationMicroseconds()
- removed most restrictions on embedded logic in user modules; user module Program objects are subject to the same restrictions as the parent Program object (if any)

- added the get_parse_options() function so that parse options in the current Program can be determined at runtime
- added the get_ex_pos() function to help with formatting exception locations where the `source` and `offset` information is present
- new methods and method variants:
  - Qore::HTTPClient::getPeerInfo()
  - Qore::HTTPClient::getSocketInfo()
  - Qore::File::getTerminalAttributes()
  - Qore::SQL::Datasource::transactionTid()
  - Qore::SQL::Datasource::currentThreadInTransaction()
  - Qore::SQL::DatasourcePool::currentThreadInTransaction()
- new location tags `"source"` and `"offset"` added for parse and runtime exceptions to allow for error-reporting to display information about files where sections of a source file are parsed; this allows both the label and line offset in the label and the file name and absolute file line position to be reported in exception information
  - new parameters added to the following methods and function to accommodate the new location information:
    - Program::parse()
    - Program::parsePending()
    - parse()
  - see Exception Hash and Call Stacks for new keys in eception and call stack information hashes
- Qore::zzz8datezzz9::format() now accepts `"us"` for microseconds (see Date Formatting Codes)
- `SmtpClient` module improvements:
  - added automatic recognition and support of the `"STARTTLS"` command when connecting to an ESMTP server; this way the class will automatically upgrade the connection to a secure TLS/SSL connection if the server supports it
  - added support for SMTP server schemes in the URL in the constructor (ex: `"esmtptls://user@password:smtp.example.com"`)
  - added support for the deprecated (but still in use) `"smtps"` scheme with a default port of 465
  - when throwing an exception when a Message cannot be sent because it is incomplete, the reason for the error is also included in the exception (previously the exception message was generic making problems with the Message object harder to debug)
- C++ API Enhancements
  - added C++ APIs to allow for Qore File and Queue object arguments to be used by modules
  - added C++ APIs for controlling openssl initialization and cleanup by the qore library
  - extended qpp to allow for parsing relative dates in qpp code for assignments/default argument values
  - made it possible to call the C++ function QoreFunction::findVariant() from threads where there is no current QoreProgram object (such as from a thread created by foreign code)
  - added APIs to allow foreign threads to be registered/deregistered as Qore threads (for example, to allow Qore code to be called in a callback in a foreign thread created by a library linked with a Qore binary module)
  - added APIs to allow for TID reservations to allow (for example) for a callback that is executed in the same foreign thread to always have the same TID
  - the old Datasource::execRaw() function with the *args* parameter was deprecated since args was ignored anyway, a new Datasource::execRaw() function was added that has no args parammeter

37.6.3 Bug Fixes in Qore

- fixed a runtime class matching bug when identical user classes were created in different Program objects, the match could fail at runtime because the wrong APIs were being used
- fixed a crashing bug in the map operator with a select expression when used with an AbstractIterator object for the list operand
- fixed a bug where the generation of internal strings for abstract method signatures tries to resolve class names that are declared out of order, which incorrectly resulted in a parse exception; the fix is to use the class name in the signature before class resolution; the class is resolved in the second stage of parsing (symbol resolution) anyway, if it can't be resolved then the changes to the Program are rolled back anyway
- a potential deadlock was fixed when calling exit() while background threads were running; it was possible for a thread to be canceled while holding a Mutex (for example) and then for another thread to deadlock trying to acquire the Mutex and therefore for the process to deadlock because pthread_mutex_lock() is not a cancellation point. The solution was to cancel all threads first, then wait half a second, then call exit()
- fixed a bug where global variables were being evaluated with strict mathematical boolean evaluation even when %perl-bool-eval was enabled (which is the default)
- fixed bug in Qore::parseBase64String() and Qore::parseBase64StringToString() when called with an empty string argument; in this case uninitialized memory was returned
- fixed runtime dynamic memory leaks in the Select From List Operator (select) and Map Operator (map) operators when used with iterators
- do thread-specific cleanup in the main thread when cleaning up/shutting down the qore library
- added additional openssl cleanup code for thread-local data and when cleaning up the qore library
- fixed a bug matching function/method variants at runtime
- fixed a race condition deleting global dynamic handlers in the `HttpServer` module
- fixed a bug where declaring an abstract method with parameters and then declaring a concrete reimplementation of the method in a child class with no parameters caused a parse-time crash
- fixed a bug where trying to dynamically call a function that does not exist results in a deadlock due to an error where a mutex is not unlocked
- fixed a bug in the Qore::Socket::sendHTTPMessage() and Qore::Socket::sendHTTPResponse() methods regarding the timeout parameter
- fixed a bug in an socket SSL error message where the method name was printed from non-string memory (used wrong ptr for the `%s` format argument)
- fixed some major crashing bugs related to reference handling; a global variable assigned a reference to a reference to a local variable would cause a crash
- reference and *reference type fixes: an error in reference type handling allowed non-reference values to be passed to builtin code expecing references which caused a crash; the reference and *reference types would accept any value type
- attempted to fix a non-reproducible bug seen on rhel5 in the Socket class where SSL errors would cause the affected thread to go into a infinite loop using 100% CPU

37.7 Qore 0.8.6.2

Release Summary

Iterator improvements and design fixes

37.7.1 Changes That Can Affect Backwards-Compatibility

Fixes for Iterator Class Design Bugs

Iterators, particular regarding the map and select operators, were implemented in a confusing and inconsistent way; even the qore documentation was incorrect, and examples were given incorrectly. The following changes will break functionality using the badly-implemented behavior of iterators before, but since the fix comes fairly soon after the introduction, hopefully this change will not cause too many problems with existing code. All users polled about the iterator changes in this release saw them as positive and desired changes to the language.

- the map and select operators' behavior was changed when used with an AbstractIterator object for the list operand; now the implied argument is the result of AbstractIterator::getValue() instead of the iterator object itself. This addresses a confusing design choice in the original iterator integration with the map and select operators
- the second boolean argument was removed from the HashIterator::constructor(hash) and HashReverse↩ Iterator::constructor(hash) methods; use the new HashPairIterator and ObjectPairIterator classes instead (Qore::zzz8hashzzz9::pairIterator() and Qore::zzz8objectzzz9::pairIterator())
- the single boolean argument was removed from Qore::zzz8hashzzz9::iterator() and Qore↩ ::zzz8objectzzz9::iterator(); use Qore::zzz8hashzzz9::pairIterator() and Qore::zzz8objectzzz9::pair↩ Iterator() instead to get the old behavior

37.7.2 Changes in Qore

- new iterator classes:
  - HashKeyIterator
  - HashKeyReverseIterator
  - HashPairIterator
  - HashPairReverseIterator
  - ObjectKeyIterator
  - ObjectKeyReverseIterator
  - ObjectPairIterator
  - ObjectPairReverseIterator
- new pseudo-methods:
  - Qore::zzz8hashzzz9::keyIterator()
  - Qore::zzz8hashzzz9::pairIterator()
  - Qore::zzz8hashzzz9::contextIterator()
  - Qore::zzz8objectzzz9::keyIterator()
  - Qore::zzz8objectzzz9::pairIterator()
  - Qore::zzz8nothingzzz9::keyIterator()
  - Qore::zzz8nothingzzz9::pairIterator()
  - Qore::zzz8nothingzzz9::contextIterator()
- the internal C++ QoreProgramHelper object has been updated to wait until all background threads in the Qore library have executed before taking the Program object out of scope; this allows for callbacks and other code that might be needed by background threads started in user modules (for example) to stay valid until the threads in the user modules also have terminated. Note that this does not affect the case when using %exec-class and an application program object goes out of scope with background threads in user modules having non-static method call references as callbacks to the application program; see Program Scope in Object-Oriented Programs Using User Modules Providing Their Own Threads for more information on this topic.

37.8 Qore 0.8.6.1

Release Summary

Major bug fixes and minor new features

37.8.1 Changes in Qore

- updated the %try-module parse directive to support a variant without an exception variable for usage in Program objects where Qore::PO_NO_TOP_LEVEL_STATEMENTS is set
- added code to raise an invalid-operation warning with the elements operator when called with a type that can never return a value with this operator
- updated the File class's internal buffer size from 4KB to 16KB which greatly improves read performance
- added new public APIs for the QoreNumberNode class to allow for proper de/serialization in external modules
- `Pop3Client` module:
  - added the Pop3Client::logPassword() methods and masked password by default in the debug log
  - updated module to v1.1
- `Mime.qm` module:
  - declared the MultiPartMessage::getMsgAndHeaders() method abstract as originally intended
  - added MultiPartMessage::parseBody() static method
  - updated module to v1.3

37.8.2 Bug Fixes in Qore

- fixed crashing bugs due to the lack of proper lvalue checks with the expression for the background operator with operators using lvalues with local variables
- fixed rounding of arbitrary-precision numeric values for display purposes when the last significant digit is just to the right of the decimal point (ex: was displaying 10.2 as "11." for example)
- fixed a race condition in static destruction of the library when a background thread calls exit() that could cause a segfault on exit
- fixed a static memory leak in Program objects when constants contain code references to functions or static methods
- fixed a bug parsing user modules; the Program context was not set properly which could lead to a crash when parsing user modules loaded from the command-line or to incorrect parse options when loaded from user Program code
- fixed a bug where the invalid-operation warning with the keys operator was not being triggered in common cases that should have triggered the warning
- `MailMessage.qm` module:
  - fixed recognizing mime messages with additional text after the version number (ex: `"Mime-Version: 1.0 (Mac OS X Mail 6.2 \(1499\))"`)
  - fixed a bug setting the content-type of message parts (this fix is now in the `Mime.qm` in the MultiPartMessage::getMsgAndHeaders() method
  - fixed multipart message parsing by using MultiPartMessage::parseBody() in the `Mime.qm` module; now also parts with subparts are parsed correctly as well
  - fixed a bug where the sender and from values were not being set properly when parsing email messages
  - updated module to v1.0.3

37.9 Qore 0.8.6

Release Summary

Major new features and a few bug fixes

37.9.1 Changes That Can Affect Backwards-Compatibility

Perl-Style Boolean Evaluation

Qore's default boolean evaluation mode was changed from strict mathematical to a more intuitive perl- (and Python-) like style. This change was implemented to address one of the oldest design bugs in Qore: strict mathematical boolean evaluation. See %perl-bool-eval for a description of the new default boolean evaluation mode.

To get the old strict mathematical boolean evaluation, use the %strict-bool-eval parse option.

An example of the change; now the following if statement block will be executed as the **if** expression is now evaluated as True:

```
my string $str = "hello";
if ($str)
    printf("Qore says hello\n");
```

Previously (i.e. with %strict-bool-eval) the **if** expression above would be evaluated as False because the string value was converted to an integer 0, however as of Qore 0.8.6 (with the default perl-bool-eval) it is True since the string is not empty; empty strings and string value `"0"` are evaluated as False.

Perhaps counterintuitively (and the reason this was changed to be the default in qore), the chance for regression errors in qore code is very small, because for all cases where the old logic could be applied (meaning excluding cases where the result was always False due to the data types or values being evaluated), the results are the same with the new logic, except for one case; the case where a string has more than one character and begins with a zero (ex: `"00"`). In this case, the old logic would always return False, because the value was first converted to an integer `0`, whereas the new logic will return True. Note that in the case of a string with a single `"0"`, both the old and new boolean logic returns False.

Basically with this option set, qore's boolean evaluation becomes like perl's and Python's, whereas any expression that has the following values is False: NOTHING, string `"0"` and empty strings, integer, float, and number `0` (zero), absolute date `1970-01-01Z` (ie the start of the epoch with an offset of 0), relative date `0s` (or any relative date with a 0 duration), NULL, empty binary objects, empty hashes, and empty lists. All other values are True.

Note

also affects the boolean(any) function

Changes in the Socket Class

The Socket class was enhanced to support timeouts with non-blocking I/O on all send operations; many Socket methods that send data were originally implemented to return an error code on error, however they would also throw exceptions if the socket were not open, so the error handling was inconsistent (exceptions versus return codes).

Additionally it was not possible to get error information at all for SSL errors if the socket was connected with SSL, which, according to Qore's socket design, should be transparent for the programmer.

For these reasons the implementation was deemed inconsistent and unintuitive; the change was to add optional timeout parameters to all send methods and to allow the methods to throw exceptions (instead of simply returning -1 and not being able to determine the cause of the error in many cases).

The following methods were updated to accept optional timeout parameters and throw exceptions on all errors rather than a return code for I/O errors:

- Qore::Socket::send()
- Qore::Socket::sendBinary()
- Qore::Socket::sendi1()
- Qore::Socket::sendi2()
- Qore::Socket::sendi4()
- Qore::Socket::sendi8()
- Qore::Socket::sendi2LSB()
- Qore::Socket::sendi4LSB()
- Qore::Socket::sendi8LSB()

New Abstract Method in AbstractIterator

The following abstract method was added:

- Qore::AbstractIterator::valid() was added (with concrete implementations in all iterator classes derived from this base class delivered with Qore); this method tells if the object is currently pointing to a valid iterator.

  For any user classes inherting AbstractIterator directly (as opposed to another concrete iterator class in Qore, where the method has already been added), a concrete implementation of this method will have to be added as well or that class will become abstract with this release of Qore.

37.9.2 New Features in Qore

Arbitrary-Precision Numeric Support

Qore now uses the `MPFR` and `GMP` libraries to provide arbitrary-precision numeric support. This type can be used for high-precision mathematics or for storing `NUMERIC` (ie `DECIMAL` or `NUMBER`) column values when retrieved from databases by Qore DBI drivers that support the new capability DBI_CAP_HAS_NUMBER_SUPPORT (previously these values would be retrieved as Qore strings in order to avoid information loss).

For more information, see the new number type, number, and Qore::zzz8numberzzz9

New CsvUtil Module

The `CsvUtil` module implements the CsvFileIterator class that allows for easy parsing of csv-like text files

%try-module Parse Directive to Handle Module Load Errors at Parse Time

The new %try-module parse directive allows for module load errors to be handled at parse time; ex:

```
%try-module($ex) some-module > 1.0
    printf("error loading module %y: %s: %s\n", $ex.arg, $ex.err, $ex.desc);
    exit(1);
%endtry
```

Abstract Class Hierarchy Improvement

As of this version of qore, concrete implementations of abstract methods no longer have to have exactly the same return type as the abstract method; it is now sufficient that the return type in the concrete method meets a compatibility test with the return type of the abstract method in the parent class.

For example the following is now valid (and `MyConcreteClass` is not abstract, whereas previously because the return types in the child class were not exact, `MyConcreteClass` would be considered abstract by qore):

```
class MyAbstractClass {
    abstract any doSomething();
    abstract *string getString();
}

class MyConcreteClass inherits MyAbstractClass {
```

```
        int doSomething() {
            return 1;
        }
        string getString() {
            return "hello";
        }
}
```

DBI Improvements

Three new DBI capabilities were implemented, including a new option API as follows:

- DBI_CAP_HAS_NUMBER_SUPPORT: DBI drivers declaring this capability can accept number values and can also return number values, if a DBI driver does not declare this capability, then number values sent for binding by value are automatically converted to float values before being sent to the driver
- DBI_CAP_HAS_OPTION_SUPPORT: this indicates that the driver supports the new option API, allowing options to be set on each connection. See the following for more information:
  - Datasource::constructor(hash): now passes options to the DBI driver if the driver supports the option API
  - Datasource::constructor(string): (new in 0.8.6) passes options to the DBI driver if the driver supports the option API
  - Datasource::getOption(string): (new in 0.8.6) returns the value of the given option if the driver supports the option API
  - Datasource::getOptionHash(): (new in 0.8.6) returns a hash of the current option values for the current connection if the driver supports the option API
  - Datasource::setOption(): (new in 0.8.6) allows options to be changed after the object is created
  - DatasourcePool::constructor(hash): now passes options to the DBI driver if the driver supports the option API
  - DatasourcePool::constructor(string): (new in 0.8.6) passes options to the DBI driver if the driver supports the option API
  - DatasourcePool::getOption(string): (new in 0.8.6) returns the value of the given option if the driver supports the option API
  - DatasourcePool::getOptionHash(): (new in 0.8.6) returns a hash of the current option values for the current connection if the driver supports the option API
  - dbi_get_driver_options(string): (new in 0.8.6) returns a hash of driver option information without values
- DBI_CAP_SERVER_TIME_ZONE: indicates that the DBI driver will convert any bound date/time values to the server's time zone before binding and also will tag date/time values retrieved from the server with the server's time zone. This capability also implies that the driver supports the new `"timezone"` option.

Socket Improvements

The Socket class was updated to support non-blocking I/O on all send methods; the following methods were updated to accept optional timeout parameters:

- Qore::Socket::send2()
- Qore::Socket::sendBinary2()
- Qore::Socket::sendHTTPMessage()
- Qore::Socket::sendHTTPResponse()

The following methods were enhanced to provide better error information when throwing exceptions:

- Qore::Socket::recvi1()
- Qore::Socket::recvi2()
- Qore::Socket::recvi4()
- Qore::Socket::recvi8()
- Qore::Socket::recvi2LSB()
- Qore::Socket::recvi4LSB()
- Qore::Socket::recvi8LSB()

- Qore::Socket::recvu1()
- Qore::Socket::recvu2()
- Qore::Socket::recvu4()
- Qore::Socket::recvu2LSB()
- Qore::Socket::recvu4LSB()

Iterator Improvements

The following improvements were made in qore to support more flexible and ubiquitous iterators:

- new iterator classes:
  - SingleValueIterator: allows single values (or any value without an iterator class) to be iterated; this provides the basis for the return type for the new base Qore::zzz8valuezzz9::iterator() method for non-container types
  - FileLineIterator: allows files to be iterated line by line
  - ObjectIterator: a generic iterator for objects
  - ObjectReverseIterator: a generic reverse iterator for objects
  - RangeIterator: a numerical sequence generator (the basis for the return type for the new xrange() function
- new pseudo-methods were added to return iterator objects based on the value type:
  - Qore::zzz8valuezzz9::iterator()
  - Qore::zzz8hashzzz9::iterator()
  - Qore::zzz8listzzz9::iterator()
  - Qore::zzz8objectzzz9::iterator()

  The base pseudo-method (Qore::zzz8valuezzz9::iterator()) ensures that any value can be iterated, and the type-specific methods ensure that the most suitable iterator for container types is returned for container values; values without an iterator class are iterated with the SingleValueIterator
- the HashIterator and HashReverseIterator classes had an additional optional boolean argument added to their constructors; if True, then the HashIterator::getValue() and HashReverseIterator::getValue() methods return a hash with the following keys: `"key"` and `"value"`, allowing for more convenient iteration with constructions that only use `getValue()` methods (such as the foreach statement); to accommodate this, two new methods were added to the HashIterator base class:
  - Qore::HashIterator::getKeyValue()
  - Qore::HashIterator::getValuePair()
- all iterator classes had copy methods added to them (ex: Qore::HashIterator::copy())
- new Python-inspired range() and xrange() functions (the latter returning a RangeIterator object to efficiently iterate large integral sequences or ranges

Text File Parsing Enhancements

The following improvements were made in qore to support more flexible file parsing:

- the Qore::ReadOnlyFile class was added as a parent class of Qore::File to allow for a more convenient API for reading files (the Qore::File class's API remains the same as it publically inherits Qore::ReadOnlyFile)
- the ReadOnlyFile::readLine() method (formerlly a method of the Qore::File class) was enhanced to accept 2 optional arguments, allowing the end of line character(s) to be stripped from the line returned, and also to allow the end of line characters to be specified. If no end of line characters are specified, then the method automatically determines the end of line characters (can be `"\n"`, `"\r"`, or `"\r\n"`; the last one only if the underlying file is not a TTY in order to avoid stalling I/O on an interactive TTY)
- the File Stat Constants were moved from the Qore::File class to the Qore::ReadOnlyFile class
- added a new FileLineIterator iterator class
- added a new optional parameter to Qore::zzz8stringzzz9::split(string, string, bool) and Qore::split(string, string, string, bool) to allow for automatic stripping unquoted fields of leading and trailing whitespace (the default is the old behavior; i.e. leave the whitespace as it is read)

- added a new TimeZone method for parsing string dates in a specific TimeZone: Qore::TimeZone↩ ::date(string, string)
- added a new function for parsing text as a boolean value: parse_boolean()
- as mentioned above, the new `CsvUtil` module was added, implementing the CsvFileIterator class that allows for easy parsing of csv-like text files

Other Improvements and Changes

- the foreach statement now iterates objects derived from AbstractIterator automatically
- added a HAVE_SYMLINK constant for the symlink() function added in qore 0.8.5
- added the SQLStatement::memberGate() method so SQLStatement objects can be dereferenced directly to a column value when iterated with SQLStatement::next(); also this method will throw exceptions when an unknown column name is used so that typos in column names can be caught (instead of being silently ignored producing hard to find bugs)
- implemented Datasource::constructor(string) and DatasourcePool::constructor(string) variants to allow for creating datasources from a string that can be parsed by Qore::SQL::parse_datasource(string) "parse↩ _datasource(string)"
- added the following new DBI-related functions:
  - dbi_get_driver_list()
  - dbi_get_driver_capability_list(string)
  - dbi_get_driver_capabilities(string)
  - dbi_get_driver_options(string)
  - parse_datasource(string)
- implemented support for `"A"` and `"a"`, (hexadecimal floating-point output) `"G"`, `"g"`, (compact floating-point output) `"F"`, (non-scientific floating-point output) and `"E"` and `"e"` (scientific/exponential floating-point output) format arguments for floats and numbers (new arbitrary-precision number type values); see String Formatting
- new pseudo-methods:
  - Qore::zzz8valuezzz9::toString()
  - Qore::zzz8valuezzz9::toInt()
  - Qore::zzz8valuezzz9::toFloat()
  - Qore::zzz8valuezzz9::toBool()
  - Qore::zzz8floatzzz9::format(string fmt)
  - Qore::zzz8intzzz9::format(string fmt)
  - Qore::zzz8stringzzz9::isDataAscii()
  - Qore::zzz8stringzzz9::isDataPrintableAscii()
  - Qore::zzz8valuezzz9::callp()
  - Qore::zzz8callrefzzz9::callp()
  - Qore::zzz8intzzz9::sign()
  - Qore::zzz8floatzzz9::sign()
- the value of the NUMBER, NUMERIC, and DECIMAL SQL Constants is now `"number"` instead of `"string"` (see also sql_binding)
- new constants:
  - M_PIn
  - MAXINT
  - MININT
- new functions:
  - range()
  - xrange()
- new methods:

  - Qore::ReadOnlyFile::isTty() and Qore::ReadOnlyFile::getFileName() (the Qore::ReadOnlyFile class was added in qore 0.8.6 otherwise made up of methods formerly belonging to the Qore::File class)
- added the %append-module-path parse directive
- user modules may now use Program objects for embedded logic; any Program objects created in a user module will have its parse options masked to be not less restrictive than the parse options in the current Program, and additionally parse options will be locked so that user module are not able to circumvent function restrictions imposed by parse options.
- updated docs to show functional restrictions tagged at the class level

37.9.3 Bug Fixes in Qore

- fixed a bug in the map operator with a select expression when the list operand is NOTHING; it was returning a list with one NOTHING element instead of NOTHING
- applied a patch by Reini Urban to allow for multi-arch builds on Debian
- fixed bugs calculating the byte offset for string searches in the c++ QoreString::index() and QoreString↩ ::rindex() functions when the offset is negative and the strings have a multi-byte character encoding (such as UTF-8)
- fixed a bug where calling an abstract method from a class where the abstract method is implemented was causing a parse error to be thrown
- fixed a bug where the wrong source code location was displayed when raising a parse exception in operator expression parse initialization for some operators
- fixed bugs in regexes in the HttpServer::addListeners() and HttpServer::addListenersWithHandler() methods (`HttpServer` module version updated to 0.3.5)
- fixed bugs handling non-blocking reads in the Socket class; the timeout setting was only enforced for the first read; subsequent reads were made as blocking reads
- fixed a bug in the Socket class when the SSL session requires renegotiation during non-blocking I/O
- File::constructor() now throws an exception if called with a tty target and %no-terminal-io is set
- fixed a bug in split with quote (Qore::zzz8stringzzz9::split(string, string, bool) and Qore::split(string, string, string, bool)) if the separator pattern was not found and the single field was not quoted either
- fixed a bug handling nested %ifdef and %ifndef blocks with %else in the inside block
- fixed a crashing due to the failure to clear the "PF_TOP_LEVEL" flag when initializing statements, this could cause temporary variables in a statement to be marked as the start of the global thread-local variable list, and then after such variables are deleted, then a crash happens when trying to access the global thread-local variable list
- fixed a crashing bug at parse time merging function lists in namespaces declared multiple times
- fixed a bug in executing user module init() closures
- fixed a bug where the qore library could crash when destroying a Program object due to a race condition in removing signal handlers managed by the Program object; the Program calls the signal handler manager to remove the signals, but the signals can be removed concurrently to the request while the Program object is iterating the signal set (ie it is modified while being iterated), which causes a crash
- added code to detect when the same namespace is declared both with and without the public keyword when defining user modules which can result in entire namespaces being silently not exported (and can be difficult to debug); now a parse exception is thrown if this happens while parsing a user module
- added code tags to File methods without side effects
- made many minor documentation fixes

37.10 Qore 0.8.5.1

Release Summary

Bugfix release

37.10.1 Bug Fixes in Qore

- fixed a race condition accessing global and closure-bound thread-local variables in multithreaded contexts
- fixed a bug in transaction management with the DatasourcePool class when used with the SQLStatement class
- fixed an error in the `MailMessage.qm` user module where mail headers requiring encoding were not encoded and those not requiring encoding were encoded with Q encoding
- fixed an error in the `Mime.qm` user module where `"_"` characters in q-encoded headers were not encoded correctly

37.11 Qore 0.8.5

Release Summary

Major new features and a few bug fixes

37.11.1 New Features in Qore

Abstract Methods and Interfaces

Qore now supports the **abstract** keyword when declaring methods; an **abstract** method has no implementation and must be implemented in child classes with the same signature for the child class to be instantiated.

Classes with **abstract** methods define interfaces; a concrete implementation of the interface is a class that inherits the class with **abstract** methods and implements all the **abstract** methods.

Abstract methods are defined with the following syntax:

```
class MyAbstractInterface {
    abstract string doSomething(int $param);
    abstract bool checkSomething(string $arg);
}
```

The following abstract classes now exist in Qore:

- AbstractDatasource
- AbstractIterator
  - AbstractQuantifiedIterator
  - AbstractBidirectionalIterator
  - AbstractQuantifiedBidirectionalIterator
- AbstractSmartLock (which was already present in Qore but now implements abstract methods)

The following new iterator classes have been added to Qore:

- HashIterator
  - HashReverseIterator
- HashListIterator
  - HashListReverseIterator
- ListHashIterator
  - ListHashReverseIterator

- ListIterator
  - ListReverseIterator
- SQLStatement (which was already present in Qore but now implements the AbstractIterator interface to allow query results to be iterated)

Classes inheriting AbstractIterator have special support so that objects can be easily iterated in the following list operators:

- Map Operator (map)
- Fold Right Operator (foldr) and Fold Left Operator (foldl)
- Select From List Operator (select)

Universal References

All restrictions on references have been removed from Qore; references to local variables may now be passed to the background operator and passed as arguments to closures.

Basically when a reference is taken of a local variable that could result in the local variable being accessed in a multi-threaded context, the variable is treated as a closure-bound local variable in the sense that it's lifetime is reference-counted, and all accesses are wrapped in a dedicated mutual-exclusion lock to ensure thread safety.

Pop3Client Module

A `Pop3Client` module has been added providing an API for communicating with `POP3` servers and retrieving email messages.

The module uses functionality provided by the new `MailMessage` module to represent email messages (and attachment data) downloaded from the server.

MailMessage Module

The `MailMessage` module provides common functionality to the `Pop3Client` and `SmtpClient` modules to represent email messages for receiving and sending, respectively. This module was created mostly from functionality removed from the `SmtpClient` and enhanced to provide support for reading email messages in the new `Pop3Client` module.

SmtpClient Module Changes

The Message and Attachment classes were removed from the `SmtpClient` module to the `MailMessage` module. Backwards-compatible definitions for the Message and Attachment classes are provided in the `SmtpClient` module to rexport the removed functionality for backwards compatibility.

Other Minor Improvements and Changes

- qpp updated to support abstract methods and multiple inheritance (+ other minor qpp enhancements)
- improved the `QOREADDRINFO-GETINFO-ERROR` exception description by adding information about the arguments passed
- added a string argument to char(softint, *string) to accept an output encoding
- added a int(string, softint) variant to parse a string as a number and give the base
- added a new parameter to parse_url() and parseURL() to allow for any [] in the hostname to be included in the `"host"` output key for indicating that the `ipv6` protocol be used
- added the following pseudo-methods:
  - Qore::zzz8valuezzz9::lsize()
  - Qore::zzz8binaryzzz9::split()
  - Qore::zzz8binaryzzz9::toMD5()
  - Qore::zzz8binaryzzz9::toSHA1()
  - Qore::zzz8binaryzzz9::toSHA224()
  - Qore::zzz8binaryzzz9::toSHA256()

  - Qore::zzz8binaryzzz9::toSHA384()
  - Qore::zzz8binaryzzz9::toSHA512()
  - Qore::zzz8datezzz9::midnight()
  - Qore::zzz8listzzz9::first()
  - Qore::zzz8listzzz9::join()
  - Qore::zzz8listzzz9::last()
  - Qore::zzz8listzzz9::lsize()
  - Qore::zzz8nothingzzz9::lsize()
  - Qore::zzz8stringzzz9::regex()
  - Qore::zzz8stringzzz9::regexExtract()
  - Qore::zzz8stringzzz9::split()
  - Qore::zzz8stringzzz9::substr()
  - Qore::zzz8stringzzz9::toMD5()
  - Qore::zzz8stringzzz9::toSHA1()
  - Qore::zzz8stringzzz9::toSHA224()
  - Qore::zzz8stringzzz9::toSHA256()
  - Qore::zzz8stringzzz9::toSHA384()
  - Qore::zzz8stringzzz9::toSHA512()
- added the `xxhash FAST algorithm` with unordered_map to Qore on supported platforms resuling in nearly 2x haster hash lookups
- added the Qore::File::isOpen() method
- added the Qore::call_pseudo() function to explicitly call a pseudo method on a value
- added the Qore::symlink() function to create symbolic links
- added Qore::TypeCodeMap and Qore::TypeNameMap to lookup type codes from type names and vice versa
- added the following functions to allow the time zone to be set per thread:
  - Qore::set_thread_tz()
  - Qore::get_thread_tz()

37.11.2 Bug Fixes in Qore

- fixed format_date() output for `"MON"` and `"DAY"`, etc
- fixed a memory leak in the parser related to parse exception handling with namespace members
- fixed an invalid assert() in module handling when an error occurs loading the module (only affected debug builds)
- tagged digest and crypto functions internally as RET_VALUE_ONLY
- do not kill TID 1 (the initial / main thread) when calling exit() in background threads as a crash can result with some 3rd party libraries that spawn their own threads on some platforms (observed on Darwin & Solaris 10 at least)
- fixed a memory bug in the new builtin function API used by modules built with qpp
- fixed memory bugs in the type system where uninitialized type pointers could be used causing a crash
- fixed a memory bug in handling "or nothing" types where a non-null pointer would be assumed to be a pointer to the type, however it could actually be a pointer to the NOTHING object, the fix was to ensure that any NOTHING objects in argument lists would be substituted with a null pointer
- fixed a bug in parse-time variant matching where an argument with parse-time type "object" would be matched as a perfect match to any parameter with any class restriction; this would cause run-time type errors if another valid class was passed that matched another variant of the method or function
- fixed a build bug that caused qore to be built twice

37.12 Qore 0.8.4

Release Summary

Major new features and changes that can affect backwards-compatibility, plus 40 bug fixes

37.12.1 Changes That Can Affect Backwards-Compatibility

Namespace Changes

Qore's internal namespace handling was nearly completely rewritten for Qore 0.8.4. This is because the old code was inefficient and applied namespaces inconsistently to Program objects.

The main change that can cause backwards-compatibility issues is that now functions are full namespace members. If no namespace is explicitly given in a function definition, the function is a member of the unnamed root namespace.

Also the distinction between builtin and user functions was removed. Internally, there is only one kind of function object, which can contain both builtin and user function variants (overloaded variants of the same function with the same name but different arguments).

All Qore builtin functions were moved to the Qore namespace.

Other namespace changes:

- loading namespaces provided by builtin modules into a Program object is now an atomic operation that may fail, if, for example, objects have already been defined in the target Program with the same name as objects provided by the builtin module. Previously this could cause undefined behavior.
- namespace lookups are now truly breadth-first as documented; previously the algorithm was depth-first (contrary to the documentation)
- namespace lookups are now done (both at parse time and runtime) with the help of symbol lookup tables for fast lookups; tables are maintained for both committed and temporary uncomitted parse symbols; this leads to up to 3x faster parsing for Qore code
- global variables are also now full namespace members, however this does not cause problems with backwards-compatibility

37.12.2 New Features in Qore

User Modules

It is now possible to develop user modules in Qore; several user modules are now included in the Qore distribution, forming Qore-language components of Qore's runtime library.

User modules delivered with Qore 0.8.4:

- `HttpServer`: a multi-threaded HTTP server implementation
- `SmtpClient`: an SMTP client library
- `TelnetClient`: a TELNET client implementation
- `Mime`: a set of MIME definitions and functions for manipulating MIME data

There are also new example programs for the above modules in the examples/ directory.

User modules are subject to Qore's functional restriction framework.

Namespace Changes

As listed above:

- global variables and functions are now full namespace members
- all builtin functions are now in the Qore namespace
- real depth-first searches are used for namespace symbols
- symbols are resolved first in the current namespace when parsing declarations/code in a namespace

The final Keyword

Classes and methods can now be declared "final" to prevent subclassing or overriding in a subclass

Pseudo Methods

Pseudo-methods are class methods that can be implemented on any value; they are also part of class hierarchy. The methods that can be executed on the value depend on the value's type, and all "pseudo-classes" inherit methods from a common base class.

For example:

```
"string".strlen()
<abf05da3>.size()
500.typeCode()
```

Are examples of pseudo-methods on literal values.

Some expensive operations such as getting the first or last key (or value) of a hash are now cheap using pseudo-methods, for example:

```
$hash.firstKey()
$hash.lastValue()
```

New Doxygen-Based Documentation

The Qore reference documentation is now generated by Doxygen, and is generated directly from the Qore sources. In fact, a new preprocessor known as "qpp" was developed for Qore 0.8.4 to facilitate and enforce doxygen documentation on Qore's runtime library (as well as abstract the relatively complex APIs used to bind C++ code to the Qore runtime library from the C++ programmer).

The documentation is more comprehensive, and corresponds much closer to the actual internal implementation since the documentation is now also contained in and directly generated from the internal C++ implementation of Qore.

For example, there is the Qore::zzz8valuezzz9::val() method. This method is implemented in the base pseudo class and is reimplemented in other pseudo-classes for other runtime data types as necessary. This method returns True if the value has a value in the same sense as Perl's boolean context evaluation. For example, if the value is a hash with no keys, it returns False; if it is a hash with keys, it returns True; if it is an empty string, it returns False; if it is a non-empty string, it returns True, etc.

LValue Handling Changes

lvalue handling was rewritten as the old implementation was ugly and subject to deadlocks (in rare corner cases).

Furthermore, medium-term, an architectural goal of Qore is to store all ints, floats, and bools internally as the basic C++ type instead of using a class wrapper for each value, which needs dynamic allocation and destruction, which takes up more memory and negatively affects execution speed.

With Qore 0.8.4, all local and global variables are stored using optimized C++ types when declared with the appropriate type restrictions; for example:

```
my int $i0;
our int $i1;
```

These declares local and global variables that can only be assigned integer values; in Qore 0.8.4 the value internally will be stored as an "int64" value (and not a dynamically-allocated QoreBigIntNode object).

The same holds for:

- int
- softint
- float
- softfloat
- bool
- softbool

Note that the optimized lvalue handling has not yet been applied to all lvalues, in particular non-static object members with declared types are not yet implemented with optimized storage; to do this requires a rewrite of Qore's API and ABI (will happen in the next major release of Qore).

This change leads to improved integer and floating-point performance and a smaller runtime memory footprint.

Runtime Optimizations

In addition to the up to 3x faster parsing (as decribed in the namespace changes above), Qore 0.8.4 contains many runtime optimizations designed to reduce the number of dynamic memory allocations performed at runtime.

The optimizations included in this version of Qore are only a half-measure compared to future changes that will necessitate a new binary Qore API.

Per-Thread Initialization

the new set_thread_init() function allows a call reference or closure to be set which will be automatically executed when new threads are started (or a new thread accesses a Program object) which can be used to transparently initialize thread-local data.

More Control Over Thread Resource Exceptions

new functions:

- throw_thread_resource_exceptions_to_mark()
- mark_thread_resources()

Allow for only thread resouces created after a certain point to be processed (for example only thread resources left after some embedded code was called)

New Socket Methods

new methods:

- Qore::Socket::upgradeClientToSSL()
- Qore::Socket::upgradeServerToSSL()

Allow upgrading an already-existing socket connection to SSL

Better Socket Error Messages

More information has been added to socket exceptions to provide better feedback when errors occur.

New Socket Event Fields

- added `"type"` and `"typename"` keys to the EVENT_HOSTNAME_RESOLVED event
- added `"type"`, `"typename"`, and `"address"` keys to the EVENT_CONNECTING event

Support For Blocking Writes in the Queue Class

Queue objects can now be used as a blocking message channel (similar to a Go channel); if a maximum size is given to the Queue constructor, then trying to write data to the Queue when it is full will block until the Queue's size goes below the maximum size; optional timeout parameters have been added to Queue methods that write to the Queue.

New Queue::clear() Method

Does just what you think it does :)

date(string, string) Improvement

added the possibility to specify microseconds when parsing dates against a mask with the date() function

New Support For ++ And – Operators With Floating-Point Lvalues

previously this would either convert the lvalue to an int or throw an exception if the lvalue could not be converted to an int due to type restrictions

Class Recognition/Compatibility Between Program Objects

The problem is that a user class created from the same source code in two different Program objects would be recognized as a different class with parameter and variable type restrictions - ie you could not declare a variable or parameter with a class type restrictions and assign it an object created from the same class source code but created in another Program object.

This problem is analogous to a similar problem with java in that classes built from the same source but from different classloaders are also recognized as different classes.

In Qore 0.8.4 a class signature is created of all public and private objects, and an SHA1 hash is maintained of the class signature, and if the class names and signatures match, then the classes are assumed to be identical, even if they have different internal class IDs (because they were created in different Program objects, for example).

New TimeZone::date(string) Method

to support creating arbitrary dates in a given TimeZone

New GetOpt::parse3() method

This method will display any errors on stderr and exit the program (which is the most typical way of handling command line errors anyway)

+= Operator Optimization For object += hash

this operation is faster in this release

New Parse Option PO_NO_MODULES

Using this option disables module loading

New Parse Option PO_NO_EMBEDDED_LOGIC

Using this option disables all dynamic parsing

New Parse Directives

- %assume-global: the opposite of %assume-local
- %old-style: the opposite of %new-style
- %require-dollar: the opposite of %allow-bare-refs
- %push-parse-options: allows parse options to be saved and restored when the current file is done parsing; very useful for include files

New Context Functions

- cx_value(): returns the value of the given key
- cx_first(): returns True if iterating the first row

- cx_last(): returns True if iterating the last row
- cx_pos(): returns the current row number (starting from 0)
- cx_total(): returns the total number of rows in the set

SOCKET-HTTP-ERROR Exception Enhancement

The invalid header info received is reported in the exception's "arg" key

Improved Parse Error Messages

Improved some parse error messages dealing with namespace and class declaration errors

Added NT_CLOSURE Constant

type code for runtime closure values

37.12.3 Bug Fixes in Qore

- fixed a race condition with Program objects when a signal handler is left active and the Program terminates
- fixed a bug in the File class where the encoding given in the constructor was ignored; if no encoding was given in the File::open*() method then the File's encoding would always be set to the default encoding, now it's set to the encoding given in the constructor (as documented)
- runtime checks have been implemented so that references to local variables cannot be passed to a closure; this would cause a runtime crash
- a fix has been made to the delete and remove operators; lists will not be extended when trying to remove/delete list elements that do not exist
- fixed some bugs showing the error location with bugs in the second stage of parsing (symbol resolution)
- apply type filters to blocks with a designated return type but no return statement
- fixed crashing bugs on some 32bit platforms where size_t was assumed to be 64 bits
- fixed a crashing bug parsing invalid %requires directives in the scanner
- fixed a bug in usleep() with relative date/time values (added a new usleep() variant to support this)
- fixed a typo in the command-line help for the qore binary with unknown parse options
- fixed HAVE_SIGNAL_HANDLING to be False if signal handling is disabled on platforms where signal handling is otherwise available
- fixed a scanner bug parsing out of line class definitions with a root-justified namespace path (ex: "class ::X::ClassName ...")
- merging code from binary modules at parse time and at runtime is now transaction-safe (before it would cause memory errors and/or a crash), now if errors are detected then an exception is raised and changes are not applied.
- fixed a crashing bug in the C++ API function QoreHashNode::setKeyValue() when the value is 0 and an exception occurs or is already active before the call is made
- fixed a bug in date parsing with a format string - off by one with integer months - added a regression test for this case
- fixed a memory error with the rethrow statement in enclosing but nested try-catch blocks
- fixed a crashing bug where qore would try to instantiate a class for a type that did not represent a class (ex: "my int $i();")
- fixed a memory leak in the softlist and *softlist type implementation

- make sure and raise a SOCKET-CLOSED error when reading a HTTP header if no data is received
- make sure and convert encodings with index() and rindex() functions if the encodings don't match
- build fix: only use a lib64 directory if the directory exists already
- raise a parse exception in the scanner if a numeric overflow occurs in literal integer values
- fixed a bug in AbstractSmartLock::lockTID()
- fixed a major crashing error in the C++ API function QoreStringNode::createAndConvertEncoding(); this function is used by the xml module when parsing XML-RPC sent in a non-UTF-8 character encoding
- fixed Qore::File::getchar() to always retrieve 1 character (even for multi-byte character encodings)
- fixed string evaluation in a boolean context to return True with floating-point numbers between -1.0 and 1.0 exclusive
- printf formatting fix: output YAML-style "null" for NOTHING with %y
- scanner fix: accept "\r" as whitespace to allow better parsing of sources with Windows EOL markers
- fixed parse-time type processing/checks for the keys, + and * operators
- foreach statement fix: unconditionally evaluate the hash when iterating as otherwise it could change during iteration which could cause a crash
- fixed another parse-time variant matching bug where the variant-matching algorithm was too aggressive and excluded possible matches at parse time which could result in a false parse-time definitive match even though a better match could be available at runtime
- fixed a static memory leak when signal handlers are left registered when the qore library terminates
- fixed static memory leaks and 1 dynamic memory leak in strmul()
- fixed a crashing bug in handling recursive constant references
- fixed a bug in the C++ API function HashIterator::deleteKey() when the node's value is NULL
- fixed time zone/DST calculations for time zone regions with DST with dates before the epoch but after the last DST transition before the epoch
- fixed a memory error where invalid source expressions referenced in a regular expression substitution expression would cause a crash (ex:

```
str =~ s/public (name)/$2/g
```

- fixed a memory error in regular expression substitution where the unconverted string (if not given in UTF-8 encoding) was used when copying source expressions to the target string
- fixed a bug where a recursive class inheritance tree would cause a crash
- fixed a bug where a static class method could not access private members of the class

Chapter 38

Bug List

Module date_and_time_functions

the "u" and "uu" format codes work with milliseconds and not microseconds since they were implemented before qore supported microsecond time resolution; this cannot be changed without breaking backwards-compatibility

currently only English month abbreviations are accepted in the date/time mask

Member Qore::format_date (string format, date dt)

there is no locale support; day and month names and abbreviations are only returned in English

Member Qore::is_readable (string path)

on Windows this function always returns True for directories; this will be fixed in a later version of Qore

Member Qore::is_writable (string path)

on Windows this function always returns False for directories; this will be fixed in a later version of Qore

Member Qore::trim (reference str, __7_ string chars)

it is not possible to trim multi-byte characters from strings using this function; each byte is treated as a character. No encoding conversions are done even if the *chars* argument has a different character encoding than the *str* argument.

Member Qore::trim (string str, __7_ string chars)

it is not possible to trim multi-byte characters from strings using this function; each byte is treated as a character. No encoding conversions are done even if the *chars* argument has a different character encoding than the *str* argument.

Member Qore::zzz8datezzz9::format (string format)

there is no locale support; day and month names and abbreviations are only returned in English

Page Signal Handling

it seems that SIGWINCH and SIGINFO cannot be handled on Darwin in Qore's dedicated signal-handling thread; the signals are never delivered to Qore's signal handling thread for some reason despite setting the internal signal masks appropriately. These signals can be handled normally in the main thread on Darwin (in other programs using traditional non-threading signal APIs), but do not work with Qore (on Darwin only when using the pthread_sigmask() and a dedicated signal-handling thread), possibly due to a bug related to signal handling and threading on Darwin.

Chapter 39

Module Index

39.1 Modules

Here is a list of all modules:

Chapter 40

Namespace Index

40.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 41

Hierarchical Index

41.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 42

Class Index

42.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 43

Module Documentation

43.1 Number Formatting Constants

Variables

- const Qore::NF_Default = QORE_NF_DEFAULT

  for the default format with a rounding heuristic to try to remove noise in insignificant digits from the binary to digital conversion of high-precision numbers
- const Qore::NF_Raw = QORE_NF_RAW

  for the raw format without the noise reduction heuristic in the NF_Default format
- const Qore::NF_Scientific = QORE_NF_SCIENTIFIC

  for the scientific format (exponential notation)

43.1.1 Detailed Description

43.1.2 Variable Documentation

43.1.2.1 const Qore::NF_Raw = QORE_NF_RAW

for the raw format without the noise reduction heuristic in the NF_Default format

note that this can be combined with binary or with NF_Scientific as well to get an exponential output without rounding

Example:

```
my string $str = $n.toString(NF_Scientific|NF_Raw);
```

43.2 Database Driver Constants

Variables

- const Qore::SQL::DSDB2 = "db2"

  for the "db2" driver
- const Qore::SQL::DSFreeTDS = "freetds"

  for the "freetds" driver
- const Qore::SQL::DSMSSQL = "freetds"

  another constant for the "freetds" driver
- const Qore::SQL::DSMySQL = "mysql"

  for the "mysql" driver
- const Qore::SQL::DSOracle = "oracle"

  for the "oracle" driver
- const Qore::SQL::DSPGSQL = "pgsql"

  for the "pgsql" driver
- const Qore::SQL::DSSQLite3 = "sqlite3"

  for the "sqlite3" driver
- const Qore::SQL::DSSybase = "sybase"

  for the "sybase" driver

43.2.1 Detailed Description

These are contants that stand for database driver names known to Qore

43.3 DBI Capability Constants

Variables

- const Qore::SQL::DBI_CAP_AUTORECONNECT = DBI_CAP_AUTORECONNECT

  Indicates that the DBI driver supports automatically/transparently reconnecting to the server if the connection is lost while not in a transaction.
- const Qore::SQL::DBI_CAP_BIND_BY_PLACEHOLDER = DBI_CAP_BIND_BY_PLACEHOLDER

  Indicates that the DBI driver supports binding placeholder buffers when executing SQL to retrieve data from queries and procedures, etc.
- const Qore::SQL::DBI_CAP_BIND_BY_VALUE = DBI_CAP_BIND_BY_VALUE

  Indicates that the DBI driver supports directly binding qore values into queries using the `%v` placeholder in the query string.
- const Qore::SQL::DBI_CAP_CHARSET_SUPPORT = DBI_CAP_CHARSET_SUPPORT

  Indicates that the DBI driver supports proper character encoding conversions.
- const Qore::SQL::DBI_CAP_EVENTS = DBI_CAP_EVENTS

  Indicates that the DBI driver supports the event API.
- const Qore::SQL::DBI_CAP_HAS_DESCRIBE = DBI_CAP_HAS_DESCRIBE

  Indicates that the DBI driver supports the describe method.
- const Qore::SQL::DBI_CAP_HAS_EXECRAW = DBI_CAP_HAS_EXECRAW

  Indicates that the DBI driver supports the Datasource::execRaw() and DatasourcePool::execRaw() methods.
- const Qore::SQL::DBI_CAP_HAS_NUMBER_SUPPORT = DBI_CAP_HAS_NUMBER_SUPPORT

  Indicates that the DBI driver supports arbitrary-precision numeric support for binding and retrieving values.
- const Qore::SQL::DBI_CAP_HAS_OPTION_SUPPORT = DBI_CAP_HAS_OPTION_SUPPORT

  Indicates that the DBI driver supports the new driver option API.
- const Qore::SQL::DBI_CAP_HAS_SELECT_ROW = DBI_CAP_HAS_SELECT_ROW

  Indicates that the DBI driver supports a native selectRow() method implementation.
- const Qore::SQL::DBI_CAP_HAS_STATEMENT = DBI_CAP_HAS_STATEMENT

  Indicates that the DBI driver supports the prepared statement interface (the SQLStatement class)
- const Qore::SQL::DBI_CAP_LOB_SUPPORT = DBI_CAP_LOB_SUPPORT

  Indicates that the DBI driver supports LOB columns (BLOBs and CLOBs, for example)
- const Qore::SQL::DBI_CAP_SERVER_TIME_ZONE = DBI_CAP_SERVER_TIME_ZONE

  Indicates that the DBI driver supports automatically converting date/time values to the server's presumed time zone (also implies that the driver supports the `"timezone"` option) and tagging date/time values with the same; this is independent from the client's current time zone setting.
- const Qore::SQL::DBI_CAP_STORED_PROCEDURES = DBI_CAP_STORED_PROCEDURES

  Indicates that the DBI driver supports stored procedure execution.
- const Qore::SQL::DBI_CAP_TIME_ZONE_SUPPORT = DBI_CAP_TIME_ZONE_SUPPORT

  Indicates that the DBI driver supports time zones in times.
- const Qore::SQL::DBI_CAP_TRANSACTION_MANAGEMENT = DBI_CAP_TRANSACTION_MANAGEMENT

  Indicates that the DBI driver supports transaction management.

43.3.1 Detailed Description

These are contants that define the meaning of bits in the capability mask such as returned by Datasource::getCapabilities()

43.4 Error Constants

Variables

- const Qore::Err::E2BIG = E2BIG

  Argument list too long.
- const Qore::Err::EACCES = EACCES

  Permission denied.
- const Qore::Err::EADDRINUSE = EADDRINUSE

  Address already in use.
- const Qore::Err::EADDRNOTAVAIL = EADDRNOTAVAIL

  Can't assign requested address.
- const Qore::Err::EAFNOSUPPORT = EAFNOSUPPORT

  Address family not supported by protocol family.
- const Qore::Err::EAGAIN = EAGAIN

  Resource temporarily unavailable.
- const Qore::Err::EALREADY = EALREADY

  Operation already in progress.
- const Qore::Err::EBADF = EBADF

  Bad file descriptor.
- const Qore::Err::EBADMSG = EBADMSG

  Bad message.
- const Qore::Err::EBUSY = EBUSY

  Device or Resource busy.
- const Qore::Err::ECHILD = ECHILD

  No child processes.
- const Qore::Err::ECONNABORTED = ECONNABORTED

  Software caused connection abort.
- const Qore::Err::ECONNREFUSED = ECONNREFUSED

  Connection refused.
- const Qore::Err::ECONNRESET = ECONNRESET

  Connection reset by peer.
- const Qore::Err::EDEADLK = EDEADLK

  Resource deadlock avoided.
- const Qore::Err::EDEADLOCK = EDEADLOCK

  Resource deadlock avoided.
- const Qore::Err::EDESTADDRREQ = EDESTADDRREQ

  Destination address required.
- const Qore::Err::EDOM = EDOM

  Numerical argument out of domain.
- const Qore::Err::EDQUOT = EDQUOT

  Disc quota exceeded.
- const Qore::Err::EEXIST = EEXIST

  File exists.
- const Qore::Err::EFAULT = EFAULT

  Bad address.
- const Qore::Err::EFBIG = EFBIG

  File too large.
- const Qore::Err::EHOSTDOWN = EHOSTDOWN

  Host is down.
- const Qore::Err::EHOSTUNREACH = EHOSTUNREACH

No route to host.

- const Qore::Err::EIDRM = EIDRM

  Identifier removed.
- const Qore::Err::EILSEQ = EILSEQ

  Illegal byte sequence.
- const Qore::Err::EINPROGRESS = EINPROGRESS

  Operation now in progress.
- const Qore::Err::EINTR = EINTR

  Interrupted system call.
- const Qore::Err::EINVAL = EINVAL

  Invalid argument.
- const Qore::Err::EIO = EIO

  Input/output error.
- const Qore::Err::EISCONN = EISCONN

  Socket is already connected.
- const Qore::Err::EISDIR = EISDIR

  Is a directory.
- const Qore::Err::ELOOP = ELOOP

  Too many levels of symbolic links.
- const Qore::Err::EMFILE = EMFILE

  Too many open files.
- const Qore::Err::EMLINK = EMLINK

  Too many links.
- const Qore::Err::EMSGSIZE = EMSGSIZE

  Message too long.
- const Qore::Err::EMULTIHOP = EMULTIHOP

  Reserved.
- const Qore::Err::ENAMETOOLONG = ENAMETOOLONG

  File name too long.
- const Qore::Err::ENETDOWN = ENETDOWN

  Network is down.
- const Qore::Err::ENETRESET = ENETRESET

  Network dropped connection on reset.
- const Qore::Err::ENETUNREACH = ENETUNREACH

  Network is unreachable.
- const Qore::Err::ENFILE = ENFILE

  Too many open files in system.
- const Qore::Err::ENOBUFS = ENOBUFS

  No buffer space available.
- const Qore::Err::ENODATA = ENODATA

  No message available on STREAM.
- const Qore::Err::ENODEV = ENODEV

  Operation not supported by device.
- const Qore::Err::ENOENT = ENOENT

  No such file or directory.
- const Qore::Err::ENOEXEC = ENOEXEC

  Exec format error.
- const Qore::Err::ENOLCK = ENOLCK

  No locks available.
- const Qore::Err::ENOLINK = ENOLINK

  Reserved.

- const Qore::Err::ENOMEM = ENOMEM

  Cannot allocate memory.
- const Qore::Err::ENOMSG = ENOMSG

  No message of desired type.
- const Qore::Err::ENOPROTOOPT = ENOPROTOOPT

  Protocol not available.
- const Qore::Err::ENOSPC = ENOSPC

  No space left on device.
- const Qore::Err::ENOSR = ENOSR

  No STREAM resources.
- const Qore::Err::ENOSTR = ENOSTR

  Not a STREAM.
- const Qore::Err::ENOSYS = ENOSYS

  Function not implemented.
- const Qore::Err::ENOTBLK = ENOTBLK

  Block device required.
- const Qore::Err::ENOTCONN = ENOTCONN

  Socket is not connected.
- const Qore::Err::ENOTDIR = ENOTDIR

  Not a directory.
- const Qore::Err::ENOTEMPTY = ENOTEMPTY

  Directory not empty.
- const Qore::Err::ENOTSOCK = ENOTSOCK

  Socket operation on non-socket.
- const Qore::Err::ENOTTY = ENOTTY

  Inappropriate ioctl for device.
- const Qore::Err::ENXIO = ENXIO

  Device not configured.
- const Qore::Err::EOPNOTSUPP = EOPNOTSUPP

  Operation not supported on socket.
- const Qore::Err::EOVERFLOW = EOVERFLOW

  Value too large to be stored in data type.
- const Qore::Err::EPERM = EPERM

  Operation not permitted.
- const Qore::Err::EPFNOSUPPORT = EPFNOSUPPORT

  Protocol family not supported.
- const Qore::Err::EPIPE = EPIPE

  Broken pipe.
- const Qore::Err::EPROTO = EPROTO

  Protocol error.
- const Qore::Err::EPROTONOSUPPORT = EPROTONOSUPPORT

  Protocol not supported.
- const Qore::Err::EPROTOTYPE = EPROTOTYPE

  Protocol wrong type for socket.
- const Qore::Err::ERANGE = ERANGE

  Result too large.
- const Qore::Err::EREMOTE = EREMOTE

  Too many levels of remote in path.
- const Qore::Err::EROFS = EROFS

  Read-only file system.
- const Qore::Err::ESHUTDOWN = ESHUTDOWN

Can't send after socket shutdown.

- const Qore::Err::ESOCKTNOSUPPORT = ESOCKTNOSUPPORT

  Socket type not supported.
- const Qore::Err::ESRCH = ESRCH

  search error
- const Qore::Err::ESTALE = ESTALE

  Stale NFS file handle.
- const Qore::Err::ETIME = ETIME

  STREAM ioctl timeout.
- const Qore::Err::ETIMEDOUT = ETIMEDOUT

  Operation timed out.
- const Qore::Err::ETOOMANYREFS = ETOOMANYREFS

  Too many references: can't splice.
- const Qore::Err::ETXTBSY = ETXTBSY

  Text file busy.
- const Qore::Err::EUSERS = EUSERS

  Too many users.
- const Qore::Err::EWOULDBLOCK = EWOULDBLOCK

  Operation would block.
- const Qore::Err::EXDEV = EXDEV

  Cross-device link.

43.4.1 Detailed Description

The constants in this group reflect the value of the host system's errno constant of the same name. If any of these constants are not defined on the current system, then they will have the value -1 in Qore.

43.5 File Open Constants

Variables

- const Qore::O_ACCMODE = O_ACCMODE

  Mask for access modes (O_RDONLY|O_WRONLY|O_RDWR)
- const Qore::O_APPEND = O_APPEND

  Open the file in append mode (append on each write)
- const Qore::O_CREAT = O_CREAT

  Create the file if it doesn't exist.
- const Qore::O_DIRECT = O_DIRECT

  direct disk access hint (0 on platforms where this is not available)
- const Qore::O_DIRECTORY = O_DIRECTORY

  must be a directory (0 on platforms where this is not available)
- const Qore::O_EXCL = O_EXCL

  Raise an error if used with O_CREAT and the file exists.
- const Qore::O_NDELAY = O_NDELAY

  synonym for O_NONBLOCK (untested with Qore; 0 on platforms where this is not available)
- const Qore::O_NOCTTY = O_NOCTTY

  don't allocate controlling tty (0 on platforms where this is not available)
- const Qore::O_NOFOLLOW = O_NOFOLLOW

  don't follow links (0 on platforms where this is not available)
- const Qore::O_NONBLOCK = O_NONBLOCK

  non-blocking I/O (untested with Qore; 0 on platforms where this is not available)
- const Qore::O_RDONLY = O_RDONLY

  Open the file read-only.
- const Qore::O_RDWR = O_RDWR

  Open for reading and writing.
- const Qore::O_SYNC = O_SYNC

  synchronized file update option (0 on platforms where this is not available)
- const Qore::O_TRUNC = O_TRUNC

  Truncate the size to zero.
- const Qore::O_WRONLY = O_WRONLY

  Open the file write-only.

43.5.1 Detailed Description

These are the possible values that can be or'ed together when calling File::open() or File::open2()

43.6 File Locking Constants

Variables

- const Qore::F_RDLCK = F_RDLCK

  Use for read-only locking.
- const Qore::F_UNLCK = F_UNLCK

  Use for unlocking a lock.
- const Qore::F_WRLCK = F_WRLCK

  Use for exclusive write locking.

43.6.1 Detailed Description

These are the possible values for the File::lock() or File::lockBlocking() methods.

Note

These contants are set to 0 on platforms where HAVE_FILE_LOCKING is False (such as native Windows ports)

43.7 File Seek Constants

Variables

- const Qore::SEEK_CUR = SEEK_CUR

  Indicates that the offset is from the current position in the file.
- const Qore::SEEK_END = SEEK_END

  Indicates that the offset is from the end of the file.
- const Qore::SEEK_SET = SEEK_SET

  Indicates that the offset is from the start of the file.

43.7.1 Detailed Description

These are the possible values for the `whence` parameter of the File::lock() and File::lockBlocking() methods

43.8 Option Constants

Variables

- const Qore::Option::HAVE_ATOMIC_OPERATIONS = bool(QORE_CONST_HAVE_ATOMIC_MACROS)

  Indicates if the Qore library supports fast atomic reference counting.
- const Qore::Option::HAVE_CLOSE_ALL_FD = bool(QORE_CONST_HAVE_CLOSE_ALL_FD)

  Indicates if the close_all_fd() function is available.
- const Qore::Option::HAVE_FILE_LOCKING = bool(QORE_CONST_HAVE_STRUCT_FLOCK)

  Indicates if the Qore library supports file locking; currently this depends on UNIX-style file locking with the fnctl() function.
- const Qore::Option::HAVE_FORK = bool(QORE_CONST_HAVE_FORK)

  Indicates if the fork() function is available.
- const Qore::Option::HAVE_GETPPID = bool(QORE_CONST_HAVE_GETPPID)

  Indicates if the getppid() function is available.
- const Qore::Option::HAVE_IS_EXECUTABLE = bool(QORE_CONST_HAVE_PWD_H)

  Indicates if the Qore library supports the is_executable() function.
- const Qore::Option::HAVE_KILL = bool(QORE_CONST_HAVE_KILL)

  Indicates if the kill() function is available.
- const Qore::Option::HAVE_LIBRARY_DEBUGGING = bool(QORE_CONST_DEBUG)

  Indicates if the Qore library has been built with debugging enabled.
- const Qore::Option::HAVE_MD2 = bool(QORE_CONST_HAVE_MD2)

  Indicates if the openssl library used to build the qore library supported the MD2 algorithm and therefore if the MD2() and MD2_bin() functions are available.
- const Qore::Option::HAVE_MDC2 = bool(QORE_CONST_HAVE_MDC2)

  Indicates if the openssl library used to build the qore library supported the MDC2 algorithm and therefore if the MDC2() and MDC2_bin() functions are available.
- const Qore::Option::HAVE_RC5 = bool(QORE_CONST_HAVE_RC5)

  Indicates if the openssl library used to build the qore library supported the RC5 encryption algorithm and therefore if the rc5_encrypt_cbc(), rc5_decrypt_cbc() and rc5_encrypt_cbc_to_string() functions are available.
- const Qore::Option::HAVE_ROUND = bool(QORE_CONST_HAVE_ROUND)

  Indicates if the round() function is available; the availability of this function depends on the presence of the C-library's round() function.
- const Qore::Option::HAVE_RUNTIME_THREAD_STACK_TRACE = bool(QORE_CONST_QORE_RUNTI↩ ME_THREAD_STACK_TRACE)

  Indicates if active thread stack tracing has been enabled as a debugging option and if the getAllThreadCallStacks() function is available.
- const Qore::Option::HAVE_SETEGID = bool(QORE_CONST_HAVE_SETEGID)

  Indicates if the setegid() function is available; the availabilty of this function depends on the system's underlying C-library.
- const Qore::Option::HAVE_SETEUID = bool(QORE_CONST_HAVE_SETEUID)

  Indicates if the seteuid() function is available; the availabilty of this function depends on the system's underlying C-library.
- const Qore::Option::HAVE_SETSID = bool(QORE_CONST_HAVE_SETSID)

  Indicates if the setsid() function is available.
- const Qore::Option::HAVE_SHA224 = bool(QORE_CONST_HAVE_SHA256)

  Indicates if the openssl library used to build the qore library supported the SHA224 algorithm and therefore if the SHA224() and SHA224_bin() functions are available.
- const Qore::Option::HAVE_SHA256 = bool(QORE_CONST_HAVE_SHA256)

  Indicates if the openssl library used to build the qore library supported the SHA256 algorithm and therefore if the SHA256() and SHA256_bin() functions are available.
- const Qore::Option::HAVE_SHA384 = bool(QORE_CONST_HAVE_SHA512)

Indicates if the openssl library used to build the qore library supported the SHA384 algorithm and therefore if the SHA384() and SHA384_bin() functions are available.

- const Qore::Option::HAVE_SHA512 = bool(QORE_CONST_HAVE_SHA512)

  Indicates if the openssl library used to build the qore library supported the SHA512 algorithm and therefore if the SHA512() and SHA512_bin() functions are available.
- const Qore::Option::HAVE_SIGNAL_HANDLING = qore(get_bool_node(QORE_CONST_HAVE_SIGNAL↩ _HANDLING && !(qore_library_options & QLO_DISABLE_SIGNAL_HANDLING)))

  Indicates if UNIX-style signal handling is available.
- const Qore::Option::HAVE_STACK_GUARD = bool(QORE_CONST_HAVE_CHECK_STACK_POS)

  Indicates if protection against stack overruns is provided.
- const Qore::Option::HAVE_STATVFS = bool(QORE_CONST_HAVE_SYS_STATVFS_H)

  Indicates if the statvfs() function is available.
- const Qore::Option::HAVE_SYMLINK = bool(QORE_CONST_HAVE_SYMLINK)

  Indicates if the symlink() function is available.
- const Qore::Option::HAVE_SYSTEM = bool(QORE_CONST_HAVE_SYSTEM)

  Indicates if the system() function is available.
- const Qore::Option::HAVE_TERMIOS = bool(QORE_CONST_HAVE_TERMIOS_H)

  Indicates if the TermIOS class is available.
- const Qore::Option::HAVE_UNIX_FILEMGT = bool(QORE_CONST_HAVE_CHOWN)

  Indicates if UNIX-style file management functionality is available (ex: chown(), Dir::chgrp(), etc)
- const Qore::Option::HAVE_UNIX_USERMGT = bool(QORE_CONST_HAVE_GETUID)

  Indicates if UNIX-style user management functionality is available (ex: getuid(), setuid(), getgid(), setgid(), etc)

43.8.1 Detailed Description

Option constants

43.8.2 Variable Documentation

43.8.2.1 const Qore::Option::HAVE_ATOMIC_OPERATIONS = bool(QORE_CONST_HAVE_ATOMIC_MACROS)

Indicates if the Qore library supports fast atomic reference counting.

Note that if this constant is False, atomic operations are supported by mutexes and are therefore slower than with native atomic reference counting operations

43.8.2.2 const Qore::Option::HAVE_FILE_LOCKING = bool(QORE_CONST_HAVE_STRUCT_FLOCK)

Indicates if the Qore library supports file locking; currently this depends on UNIX-style file locking with the fnctl() function.

Note

Currently this function is only available when running on UNIX or UNIX-like platforms

43.8.2.3 const Qore::Option::HAVE_FORK = bool(QORE_CONST_HAVE_FORK)

Indicates if the fork() function is available.

Note

This constant is always False on native Windows ports

43.8.2.4 const Qore::Option::HAVE_GETPPID = bool(QORE_CONST_HAVE_GETPPID)

Indicates if the getppid() function is available.

Note

This constant is always False on native Windows ports

43.8.2.5 const Qore::Option::HAVE_IS_EXECUTABLE = bool(QORE_CONST_HAVE_PWD_H)

Indicates if the Qore library supports the is_executable() function.

Note

Currently this function is only available when running on UNIX or UNIX-like platforms

43.8.2.6 const Qore::Option::HAVE_KILL = bool(QORE_CONST_HAVE_KILL)

Indicates if the kill() function is available.

Note

This constant is always False on native Windows ports

43.8.2.7 const Qore::Option::HAVE_SETEGID = bool(QORE_CONST_HAVE_SETEGID)

Indicates if the setegid() function is available; the availabilty of this function depends on the system's underlying C-library.

Note

This constant is always False on native Windows ports

43.8.2.8 const Qore::Option::HAVE_SETEUID = bool(QORE_CONST_HAVE_SETEUID)

Indicates if the seteuid() function is available; the availabilty of this function depends on the system's underlying C-library.

Note

This constant is always False on native Windows ports

43.8.2.9 const Qore::Option::HAVE_SETSID = bool(QORE_CONST_HAVE_SETSID)

Indicates if the setsid() function is available.

Note

This constant is always False on native Windows ports

43.8.2.10 const Qore::Option::HAVE_SIGNAL_HANDLING = qore(get_bool_node(QORE_CONST_HAVE_SIGNAL_HANDLING && !(qore_library_options & QLO_DISABLE_SIGNAL_HANDLING)))

Indicates if UNIX-style signal handling is available.

If this constant is False, then the set_signal_handler() and remove_signal_handler() functions are not available.

Note

This constant is always False on native Windows ports and is also False if the qore library was initialized with signals disabled (such as with `qore -b` or `qore -disable-signals`)

See also

Signal Handling

43.8.2.11 const Qore::Option::HAVE_STATVFS = bool(QORE_CONST_HAVE_SYS_STATVFS_H)

Indicates if the statvfs() function is available.

Note

This constant is always False on native Windows ports

43.8.2.12 const Qore::Option::HAVE_SYMLINK = bool(QORE_CONST_HAVE_SYMLINK)

Indicates if the symlink() function is available.

Note

This constant is always False on native Windows ports

Since

Qore 0.8.6 although the symlink() function was added in 0.8.5

43.8.2.13 const Qore::Option::HAVE_TERMIOS = bool(QORE_CONST_HAVE_TERMIOS_H)

Indicates if the TermIOS class is available.

Note

This constant is always False on native Windows ports

43.8.2.14 const Qore::Option::HAVE_UNIX_FILEMGT = bool(QORE_CONST_HAVE_CHOWN)

Indicates if UNIX-style file management functionality is available (ex: chown(), Dir::chgrp(), etc)

Note

This constant is always False on native Windows ports

43.8.2.15 const Qore::Option::HAVE_UNIX_USERMGT = bool(QORE_CONST_HAVE_GETUID)

Indicates if UNIX-style user management functionality is available (ex: getuid(), setuid(), getgid(), setgid(), etc)

Note

This constant is always False on native Windows ports

43.9 Parse Option Constants

Variables

- const Qore::PO_ALLOW_BARE_REFS = PO_ALLOW_BARE_REFS

  Prohibits the use of the '$' character in variable names, method calls, and object member references.
- const Qore::PO_ASSUME_LOCAL = PO_ASSUME_LOCAL

  Assume local variable scope when variables are first referenced if no ***my*** *or* ***our*** *is present.*
- const Qore::PO_DEFAULT = PO_DEFAULT

  This option is the empty option, meaning no options are set.
- const Qore::PO_FREE_OPTIONS = PO_FREE_OPTIONS

  mask of options that have no effect on code access or code safety but just affect programming style
- const Qore::PO_IN_MODULE = PO_IN_MODULE

  Only set by the system when in a user module Program.
- const Qore::PO_LOCKDOWN = PO_LOCKDOWN

  Sets very restrictive access; this restriction is designed to allow code to only execute logic, no I/O, no threading, no external access.
- const Qore::PO_LOCK_WARNINGS = PO_LOCK_WARNINGS

  Disallows changes to the warning mask.
- const Qore::PO_NEW_STYLE = PO_NEW_STYLE

  Set a more C++ or Java type programming style; prohibits usage of the "$" character and also assumes local variable scope without ***my****.*
- const Qore::PO_NO_CHILD_PO_RESTRICTIONS = PO_NO_CHILD_PO_RESTRICTIONS

  Allows child program objects to have fewer parse restrictions (i.e. more capabilies) than the parent object.
- const Qore::PO_NO_CLASS_DEFS = PO_NO_CLASS_DEFS

  Disallows class definitions.
- const Qore::PO_NO_CONSTANT_DEFS = PO_NO_CONSTANT_DEFS

  Disallows constant definitions.
- const Qore::PO_NO_DATABASE = PO_NO_DATABASE

  Disallows access to database functionality.
- const Qore::PO_NO_EMBEDDED_LOGIC = PO_NO_EMBEDDED_LOGIC

  Prohibits embedded logic from being used.
- const Qore::PO_NO_EXTERNAL_ACCESS = PO_NO_EXTERNAL_ACCESS

  Prohibits any external access.
- const Qore::PO_NO_EXTERNAL_INFO = PO_NO_EXTERNAL_INFO

  Disallows access to functionality that provides information about the computing environment.
- const Qore::PO_NO_EXTERNAL_PROCESS = PO_NO_EXTERNAL_PROCESS

  Disallows any access to external processes (with system(), backquote(), exec(), etc)
- const Qore::PO_NO_FILESYSTEM = PO_NO_FILESYSTEM

  Disallows access to the filesystem.
- const Qore::PO_NO_GLOBAL_VARS = PO_NO_GLOBAL_VARS

  Disallows the use of global variables.
- const Qore::PO_NO_GUI = PO_NO_GUI

  Disallows access to functionality that draws graphics to the display.
- const Qore::PO_NO_INHERIT_GLOBAL_VARS = PO_NO_INHERIT_GLOBAL_VARS

  Precludes global variables from being inherited into the new Program object.
- const Qore::PO_NO_INHERIT_USER_FUNC_VARIANTS = PO_NO_INHERIT_USER_FUNC_VARIANTS

  Precludes public user function variants from being inherited into the new Program object.
- const Qore::PO_NO_IO = PO_NO_IO

  Prohibits all terminal and file I/O and GUI operations.
- const Qore::PO_NO_LOCALE_CONTROL = PO_NO_LOCALE_CONTROL

Disallows access to functionality that can change locale parameters.

- const Qore::PO_NO_MODULES = PO_NO_MODULES

  Disallows loading modules with the %requires directive or at runtime with load_module()
- const Qore::PO_NO_NAMESPACE_DEFS = PO_NO_NAMESPACE_DEFS

  Disallows new namespace definitions.
- const Qore::PO_NO_NETWORK = PO_NO_NETWORK

  Disallows access to network functionality.
- const Qore::PO_NO_NEW = PO_NO_NEW

  Disallows use of the new operator.
- const Qore::PO_NO_PROCESS_CONTROL = PO_NO_PROCESS_CONTROL

  Disallows access to functions that would affect the current process (exit(), exec(), fork(), etc)
- const Qore::PO_NO_SUBROUTINE_DEFS = PO_NO_SUBROUTINE_DEFS

  Disallows subroutine (function) definitions.
- const Qore::PO_NO_SYSTEM_CLASSES = PO_NO_SYSTEM_CLASSES

  Prohibits system classes from being imported into the new Program object.
- const Qore::PO_NO_SYSTEM_FUNC_VARIANTS = PO_NO_SYSTEM_FUNC_VARIANTS

  Prohibits builtin/system function variants from being imported into the new Program object.
- const Qore::PO_NO_TERMINAL_IO = PO_NO_TERMINAL_IO

  Disallows access to reading from and/or writing to the terminal.
- const Qore::PO_NO_THREADS = PO_NO_THREADS

  Prohibits access to all threading information.
- const Qore::PO_NO_THREAD_CLASSES = PO_NO_THREAD_CLASSES

  Disallows access to any thread classes.
- const Qore::PO_NO_THREAD_CONTROL = PO_NO_THREAD_CONTROL

  Disallows access to any thread-control functions and thread-relevant statements and operators (for example the background operator and the thread_exit statement)
- const Qore::PO_NO_THREAD_INFO = PO_NO_THREAD_INFO

  Disallows access to functionality that provides information about threading.
- const Qore::PO_NO_TOP_LEVEL_STATEMENTS = PO_NO_TOP_LEVEL_STATEMENTS

  Disallows top level code.
- const Qore::PO_NO_USER_CLASSES = PO_NO_USER_CLASSES

  Prohibits user classes from being imported into the new Program object.
- const Qore::PO_POSITIVE_OPTIONS = PO_POSITIVE_OPTIONS

  mask of all parse options allowing for more freedom (instead of less)
- const Qore::PO_REQUIRE_OUR = PO_REQUIRE_OUR

  Requires global variables to be declared with our before use.
- const Qore::PO_REQUIRE_PROTOTYPES = PO_REQUIRE_PROTOTYPES

  Requires all function and method parameters and return types to have type declarations.
- const Qore::PO_REQUIRE_TYPES = PO_REQUIRE_TYPES

  Requires all function and method parameters, return types, variables, and object members to have type declarations.
- const Qore::PO_STRICT_ARGS = PO_STRICT_ARGS

  Prohibits access to builtin functions and methods flagged with RT_NOOP and also causes errors to be raised if excess arguments are given to functions that do not access excess arguments.
- const Qore::PO_STRICT_BOOLEAN_EVAL = PO_STRICT_BOOLEAN_EVAL

  Sets strict mathematical boolean evaluation runtime mode (the qore default prior to v0.8.6)

43.9.1 Detailed Description

These are the possible values that can be or'ed together to set Program parse options

43.9.2 Variable Documentation

43.9.2.1 const Qore::PO_ALLOW_BARE_REFS = PO_ALLOW_BARE_REFS

Prohibits the use of the '$' character in variable names, method calls, and object member references.

This option should probably be named "PO_REQUIRE_BARE_REFS" as it is an error to use the "$" character when this option is set

See also

%allow-bare-refs

43.9.2.2 const Qore::PO_ASSUME_LOCAL = PO_ASSUME_LOCAL

Assume local variable scope when variables are first referenced if no **my** or **our** is present.

This option is set implicitly with PO_NEW_STYLE

See also

%assume-local

43.9.2.3 const Qore::PO_FREE_OPTIONS = PO_FREE_OPTIONS

mask of options that have no effect on code access or code safety but just affect programming style

made up of PO_ALLOW_BARE_REFS | PO_ASSUME_LOCAL

43.9.2.4 const Qore::PO_IN_MODULE = PO_IN_MODULE

Only set by the system when in a user module Program.

Note

This option cannot be set manually or an exception will be raised

43.9.2.5 const Qore::PO_LOCK_WARNINGS = PO_LOCK_WARNINGS

Disallows changes to the warning mask.

See also

%lock-warnings

43.9.2.6 const Qore::PO_LOCKDOWN = PO_LOCKDOWN

Sets very restrictive access; this restriction is designed to allow code to only execute logic, no I/O, no threading, no external access.

made up of PO_NO_EXTERNAL_ACCESS | PO_NO_THREADS | PO_NO_IO

See also

%lockdown

43.9.2.7 const Qore::PO_NEW_STYLE = PO_NEW_STYLE

Set a more C++ or Java type programming style; prohibits usage of the `"$"` character and also assumes local variable scope without **my**.

made up of PO_ALLOW_BARE_REFS | PO_ASSUME_LOCAL

See also

%new-style

43.9.2.8 const Qore::PO_NO_CHILD_PO_RESTRICTIONS = PO_NO_CHILD_PO_RESTRICTIONS

Allows child program objects to have fewer parse restrictions (i.e. more capabilies) than the parent object.

See also

%no-child-restrictions

43.9.2.9 const Qore::PO_NO_CLASS_DEFS = PO_NO_CLASS_DEFS

Disallows class definitions.

See also

%no-class-defs

43.9.2.10 const Qore::PO_NO_CONSTANT_DEFS = PO_NO_CONSTANT_DEFS

Disallows constant definitions.

See also

%no-constant-defs

43.9.2.11 const Qore::PO_NO_DATABASE = PO_NO_DATABASE

Disallows access to database functionality.

See also

%no-database

43.9.2.12 const Qore::PO_NO_EXTERNAL_ACCESS = PO_NO_EXTERNAL_ACCESS

Prohibits any external access.

made up of PO_NO_PROCESS_CONTROL | PO_NO_NETWORK | PO_NO_FILESYSTEM | PO_NO_DATABASE | PO_NO_EXTERNAL_INFO | PO_NO_EXTERNAL_PROCESS

See also

%no-external-access

43.9.2.13 const Qore::PO_NO_EXTERNAL_INFO = PO_NO_EXTERNAL_INFO

Disallows access to functionality that provides information about the computing environment.

See also

%no-external-info

43.9.2.14 const Qore::PO_NO_EXTERNAL_PROCESS = PO_NO_EXTERNAL_PROCESS

Disallows any access to external processes (with system(), backquote(), exec(), etc)

See also

%no-external-process

43.9.2.15 const Qore::PO_NO_FILESYSTEM = PO_NO_FILESYSTEM

Disallows access to the filesystem.

See also

%no-filesystem

43.9.2.16 const Qore::PO_NO_GLOBAL_VARS = PO_NO_GLOBAL_VARS

Disallows the use of global variables.

See also

%no-global-vars

43.9.2.17 const Qore::PO_NO_GUI = PO_NO_GUI

Disallows access to functionality that draws graphics to the display.

See also

PO_NO_TERMINAL_IO
%no-gui

43.9.2.18 const Qore::PO_NO_IO = PO_NO_IO

Prohibits all terminal and file I/O and GUI operations.

made up of PO_NO_GUI | PO_NO_TERMINAL_IO | PO_NO_FILESYSTEM | PO_NO_NETWORK | PO_NO_D↩ATABASE

See also

%no-io

43.9.2.19 const Qore::PO_NO_LOCALE_CONTROL = PO_NO_LOCALE_CONTROL

Disallows access to functionality that can change locale parameters.

See also

%no-locale-control

43.9.2.20 const Qore::PO_NO_MODULES = PO_NO_MODULES

Disallows loading modules with the %requires directive or at runtime with load_module()

See also

%no-modules

43.9.2.21 const Qore::PO_NO_NAMESPACE_DEFS = PO_NO_NAMESPACE_DEFS

Disallows new namespace definitions.

See also

%no-namespace-defs

43.9.2.22 const Qore::PO_NO_NETWORK = PO_NO_NETWORK

Disallows access to network functionality.

See also

%no-network

43.9.2.23 const Qore::PO_NO_NEW = PO_NO_NEW

Disallows use of the new operator.

Note that objects can still be instantiated with syntax like:

```
my Mutex $m();
```

See also

%no-new

43.9.2.24 const Qore::PO_NO_PROCESS_CONTROL = PO_NO_PROCESS_CONTROL

Disallows access to functions that would affect the current process (exit(), exec(), fork(), etc)

See also

%no-process-control

43.9.2.25 const Qore::PO_NO_SUBROUTINE_DEFS = PO_NO_SUBROUTINE_DEFS

Disallows subroutine (function) definitions.

See also

%no-subroutine-defs

43.9.2.26 const Qore::PO_NO_TERMINAL_IO = PO_NO_TERMINAL_IO

Disallows access to reading from and/or writing to the terminal.

See also

%no-terminal-io

43.9.2.27 const Qore::PO_NO_THREAD_CLASSES = PO_NO_THREAD_CLASSES

Disallows access to any thread classes.

See also

%no-thread-classes

43.9.2.28 const Qore::PO_NO_THREAD_CONTROL = PO_NO_THREAD_CONTROL

Disallows access to any thread-control functions and thread-relevant statements and operators (for example the background operator and the thread_exit statement)

See also

%no-thread-control

43.9.2.29 const Qore::PO_NO_THREAD_INFO = PO_NO_THREAD_INFO

Disallows access to functionality that provides information about threading.

See also

%no-thread-info

43.9.2.30 const Qore::PO_NO_THREADS = PO_NO_THREADS

Prohibits access to all threading information.

made up of PO_NO_THREAD_CONTROL | PO_NO_THREAD_CLASSES | PO_NO_THREAD_INFO

See also

%no-threads

43.9.2.31 const Qore::PO_NO_TOP_LEVEL_STATEMENTS = PO_NO_TOP_LEVEL_STATEMENTS

Disallows top level code.

See also

%no-top-level

43.9.2.32 const Qore::PO_POSITIVE_OPTIONS = PO_POSITIVE_OPTIONS

mask of all parse options allowing for more freedom (instead of less)

made up of only PO_NO_CHILD_PO_RESTRICTIONS

43.9.2.33 const Qore::PO_REQUIRE_OUR = PO_REQUIRE_OUR

Requires global variables to be declared with our before use.

See also

%require-our

43.9.2.34 const Qore::PO_REQUIRE_PROTOTYPES = PO_REQUIRE_PROTOTYPES

Requires all function and method parameters and return types to have type declarations.

However, variable declarations and object members are not required to have type declarations

See also

PO_REQUIRE_TYPES
%require-prototypes

43.9.2.35 const Qore::PO_REQUIRE_TYPES = PO_REQUIRE_TYPES

Requires all function and method parameters, return types, variables, and object members to have type declarations.

Note

This option also implies PO_STRICT_ARGS

See also

PO_REQUIRE_PROTOTYPES
%require-types

43.9.2.36 const Qore::PO_STRICT_ARGS = PO_STRICT_ARGS

Prohibits access to builtin functions and methods flagged with RT_NOOP and also causes errors to be raised if excess arguments are given to functions that do not access excess arguments.

This option is set implicitly with PO_REQUIRE_TYPES

See also

%strict-args

43.9.2.37 const Qore::PO_STRICT_BOOLEAN_EVAL = PO_STRICT_BOOLEAN_EVAL

Sets strict mathematical boolean evaluation runtime mode (the qore default prior to v0.8.6)

See also

%strict-bool-eval

Since

Qore 0.8.6

43.10 Warning Constants

Variables

- const Qore::WARN_ALL = QP_WARN_ALL

  Enables all warnings.
- const Qore::WARN_CALL_WITH_TYPE_ERRORS = QP_WARN_CALL_WITH_TYPE_ERRORS

  Enables warnings when the parser determines that the argument types of a function or method call are such that the operation is guaranteed to produce a constant value.
- const Qore::WARN_DEFAULT = QP_WARN_DEFAULT

  The default warning mask.
- const Qore::WARN_DEPRECATED = QP_WARN_DEPRECATED

  Enables a warning when deprecated code is used.
- const Qore::WARN_DUPLICATE_BLOCK_VARS = QP_WARN_DUPLICATE_BLOCK_VARS

  Enables a warning when a program declares a local variable more than once in the same block; note that this is not a warning but rather an error when assume-local or new-style parse options are set.
- const Qore::WARN_DUPLICATE_GLOBAL_VARS = QP_WARN_DUPLICATE_GLOBAL_VARS

  Indicates that the embedded code has declared the same global variable more than once.
- const Qore::WARN_DUPLICATE_HASH_KEY = QP_WARN_DUPLICATE_HASH_KEY

  Enables a warning when an immediate hash is declared and at least one of the keys is repeated.
- const Qore::WARN_DUPLICATE_LOCAL_VARS = QP_WARN_DUPLICATE_LOCAL_VARS

  Enables a warning when a local variable with the same name is declared in a subblock (ie another local variable with the same name is reachable in the same lexical scope); note that this warning can raise false positives if the programmer is used to redeclaring the same variable names in subblocks.
- const Qore::WARN_EXCESS_ARGS = QP_WARN_EXCESS_ARGS

  Enables a warning when a function or method call is made with more arguments than are used by the function or method.
- const Qore::WARN_INVALID_OPERATION = QP_WARN_INVALID_OPERATION

  Indicates that the embedded code performs some operation that is guaranteed to produce no result (for example, using the [] operator on an integer value)
- const Qore::WARN_MODULES = QP_WARN_MODULES

  The default warning mask for user modules.
- const Qore::WARN_NONE = QP_WARN_NONE

  Represents no warning.
- const Qore::WARN_NONEXISTENT_METHOD_CALL = QP_WARN_NONEXISTENT_METHOD_CALL

  Indicates that the embedded code is calling an unknown method in a class.
- const Qore::WARN_RETURN_VALUE_IGNORED = QP_WARN_RETURN_VALUE_IGNORED

  Enables a warning when a function or method call is made with no side effects and the return value is ignored.
- const Qore::WARN_UNDECLARED_VAR = QP_WARN_UNDECLARED_VAR

  Indicates that the embedded code referenced an undeclared variable that will be assumed to be a global variable.
- const Qore::WARN_UNKNOWN_WARNING = QP_WARN_UNKNOWN_WARNING

  Indicates that the embedded code tried to enable or disable an unknown warning.
- const Qore::WARN_UNREACHABLE_CODE = QP_WARN_UNREACHABLE_CODE

  Indicates that code cannot be reached (for example; code in the same local block after an unconditional return or thread_exit statement)
- const Qore::WARN_UNREFERENCED_VARIABLE = QP_WARN_UNREFERENCED_VARIABLE

  This warning is raised when a variable is declared in a block but never referenced.
- const Qore::WARN_WARNING_MASK_UNCHANGED = QP_WARN_WARNING_MASK_UNCHANGED

  This warning means that the embedded code tried to change the warning mask, but it was locked, so the warning mask was actually unchanged.

43.10.1 Detailed Description

These are the possible values that can be or'ed together to set the Program warning mask

43.10.2 Variable Documentation

43.10.2.1 const Qore::WARN_CALL_WITH_TYPE_ERRORS = QP_WARN_CALL_WITH_TYPE_ERRORS

Enables warnings when the parser determines that the argument types of a function or method call are such that the operation is guaranteed to produce a constant value.

See also

call-with-type-errors

43.10.2.2 const Qore::WARN_DEFAULT = QP_WARN_DEFAULT

The default warning mask.

This warning is made up of the following values combined with binary or:

- WARN_UNKNOWN_WARNING
- WARN_UNREACHABLE_CODE
- WARN_NONEXISTENT_METHOD_CALL
- WARN_INVALID_OPERATION
- WARN_CALL_WITH_TYPE_ERRORS
- WARN_RETURN_VALUE_IGNORED
- WARN_DEPRECATED
- WARN_DUPLICATE_HASH_KEY
- WARN_DUPLICATE_BLOCK_VARS

43.10.2.3 const Qore::WARN_DEPRECATED = QP_WARN_DEPRECATED

Enables a warning when deprecated code is used.

See also

"Deprecated" Functions

43.10.2.4 const Qore::WARN_DUPLICATE_BLOCK_VARS = QP_WARN_DUPLICATE_BLOCK_VARS

Enables a warning when a program declares a local variable more than once in the same block; note that this is not a warning but rather an error when assume-local or new-style parse options are set.

See also

duplicate-block-vars

43.10.2.5 const Qore::WARN_DUPLICATE_GLOBAL_VARS = QP_WARN_DUPLICATE_GLOBAL_VARS

Indicates that the embedded code has declared the same global variable more than once.

See also

duplicate-global-vars

43.10.2.6 const Qore::WARN_DUPLICATE_HASH_KEY = QP_WARN_DUPLICATE_HASH_KEY

Enables a warning when an immediate hash is declared and at least one of the keys is repeated.

See also

duplicate-hash-key

43.10.2.7 const Qore::WARN_DUPLICATE_LOCAL_VARS = QP_WARN_DUPLICATE_LOCAL_VARS

Enables a warning when a local variable with the same name is declared in a subblock (ie another local variable with the same name is reachable in the same lexical scope); note that this warning can raise false positives if the programmer is used to redeclaring the same variable names in subblocks.

See also

duplicate-local-vars and WARN_DUPLICATE_BLOCK_VARS

43.10.2.8 const Qore::WARN_EXCESS_ARGS = QP_WARN_EXCESS_ARGS

Enables a warning when a function or method call is made with more arguments than are used by the function or method.

See also

excess-args

43.10.2.9 const Qore::WARN_INVALID_OPERATION = QP_WARN_INVALID_OPERATION

Indicates that the embedded code performs some operation that is guaranteed to produce no result (for example, using the [] operator on an integer value)

See also

invalid-operation

43.10.2.10 const Qore::WARN_MODULES = QP_WARN_MODULES

The default warning mask for user modules.

This warning is made up of the following values combined with binary or:

- WARN_UNREACHABLE_CODE
- WARN_NONEXISTENT_METHOD_CALL

- WARN_INVALID_OPERATION
- WARN_CALL_WITH_TYPE_ERRORS
- WARN_RETURN_VALUE_IGNORED
- WARN_DUPLICATE_HASH_KEY
- WARN_DUPLICATE_BLOCK_VARS

43.10.2.11 const Qore::WARN_NONEXISTENT_METHOD_CALL = QP_WARN_NONEXISTENT_METHOD_CALL

Indicates that the embedded code is calling an unknown method in a class.

This warning may generate false positives; it may be vaild operation if the calling method is only called from a derived class that actually implements the method. In this case, the cast<> operator can be used to eliminate the warning.

See also

non-existent-method-call

43.10.2.12 const Qore::WARN_RETURN_VALUE_IGNORED = QP_WARN_RETURN_VALUE_IGNORED

Enables a warning when a function or method call is made with no side effects and the return value is ignored.

See also

return-value-ignored

43.10.2.13 const Qore::WARN_UNDECLARED_VAR = QP_WARN_UNDECLARED_VAR

Indicates that the embedded code referenced an undeclared variable that will be assumed to be a global variable.

See also

undeclared-var

43.10.2.14 const Qore::WARN_UNKNOWN_WARNING = QP_WARN_UNKNOWN_WARNING

Indicates that the embedded code tried to enable or disable an unknown warning.

See also

unknown-warning

43.10.2.15 const Qore::WARN_UNREACHABLE_CODE = QP_WARN_UNREACHABLE_CODE

Indicates that code cannot be reached (for example; code in the same local block after an unconditional return or thread_exit statement)

See also

unreachable-code

43.10.2.16 const Qore::WARN_UNREFERENCED_VARIABLE = QP_WARN_UNREFERENCED_VARIABLE

This warning is raised when a variable is declared in a block but never referenced.

See also

unreferenced-variable

43.10.2.17 const Qore::WARN_WARNING_MASK_UNCHANGED = QP_WARN_WARNING_MASK_UNCHANGED

This warning means that the embedded code tried to change the warning mask, but it was locked, so the warning mask was actually unchanged.

See also

warning-mask-unchanged

43.11 Type Code Constants

Variables

- const Qore::NT_BINARY = NT_BINARY

  type code for binary values
- const Qore::NT_BOOLEAN = NT_BOOLEAN

  type code for boolean values
- const Qore::NT_CALLREF = NT_FUNCREF

  type code for call references
- const Qore::NT_CLOSURE = NT_RUNTIME_CLOSURE

  type code for closures
- const Qore::NT_DATE = NT_DATE

  type code for date/time values
- const Qore::NT_FLOAT = NT_FLOAT

  type code for float values
- const Qore::NT_HASH = NT_HASH

  type code for hash values
- const Qore::NT_INT = NT_INT

  type code for integer values
- const Qore::NT_LIST = NT_LIST

  type code for list values
- const Qore::NT_NOTHING = NT_NOTHING

  type code for no value (NOTHING)
- const Qore::NT_NULL = NT_NULL

  type code for NULL
- const Qore::NT_NUMBER = NT_NUMBER

  type code for number values
- const Qore::NT_OBJECT = NT_OBJECT

  type code for objects
- const Qore::NT_STRING = NT_STRING

  type code for string values

43.11.1 Detailed Description

Type code constants as returned by Qore::zzz8valuezzz9::typeCode()

43.12 Type Code Map Constants

Variables

- const Qore::TypeCodeMap

  type code map, looks up type names from type code values
- const Qore::TypeNameMap

  type name map, looks up type codes from type names

43.12.1 Detailed Description

Type code map constants to look up type codes (as returned by Qore::zzz8valuezzz9::typeCode()) from type names (as returned by Qore::zzz8valuezzz9::type()) and vice-versa.

43.13 Boolean Constants

Variables

- const Qore::False = bool(false)

  logical False
- const Qore::True = bool(true)

  logical True

43.13.1 Detailed Description

43.14 NULL and NOTHING Constants

Variables

- const Qore::NOTHING = qore(&Nothing)

  a constant representing the lack of a value
- const Qore::NULL = qore(&Null)

  logical False

43.14.1 Detailed Description

43.15 Exception Type Constants

Variables

- const Qore::ET_System = "System"

  Exception type code system exceptions (thrown in internal Qore code or in modules)
- const Qore::ET_User = "User"

  Exception type for user exceptions (thrown by the throw statement)

43.15.1 Detailed Description

43.16 Call Type Constants

Variables

- const Qore::CT_Builtin = CT_BUILTIN

  Call type for builtin code.
- const Qore::CT_NewThread = CT_NEWTHREAD

  Call type for the start of a new thread by the background operator.
- const Qore::CT_Rethrow = CT_RETHROW

  Call type for an exception thrown by the rethrow statement.
- const Qore::CT_User = CT_USER

  Call type for user code.

43.16.1 Detailed Description

43.17 System and Build Constants

Variables

- const Qore::Build = qore(new QoreBigIntNode(qore_build_number))

  The integer Qore build number.
- const Qore::BuildHost = qore(new QoreStringNode(qore_build_host))

  The host name of the host used to build the Qore library.
- const Qore::CFLAGS = qore(new QoreStringNode(qore_cflags))

  A string giving the C++ compiler flags used to build Qore.
- const Qore::Compiler = qore(new QoreStringNode(qore_cplusplus_compiler))

  A string giving the C++ compiler used to build Qore.
- const Qore::LDFLAGS = qore(new QoreStringNode(qore_ldflags))

  A string giving the linker flags used to build Qore.
- const Qore::MACHINE_MSB = bool(Q_MACHINE_MSB)

  True if the current machine uses `big-endian or MSB byte order` *or False if the current machine uses* `little-endian or LSB byte order`
- const Qore::PlatformCPU = qore(new QoreStringNode(TARGET_ARCH))

  The string for the platform's CPU architecture.
- const Qore::PlatformOS = qore(new QoreStringNode(TARGET_OS))

  A string giving the platform operating-system name.
- const Qore::VersionMajor = qore(new QoreBigIntNode(qore_version_major))

  The integer Qore major version number.
- const Qore::VersionMinor = qore(new QoreBigIntNode(qore_version_minor))

  The integer Qore minor version number.
- const Qore::VersionString = qore(new QoreStringNode(qore_version_string))

  The full Qore version string.
- const Qore::VersionSub = qore(new QoreBigIntNode(qore_version_sub))

  The integer Qore sub version number.

43.17.1 Detailed Description

43.18 Event Source Constants

Variables

- const Qore::SOURCE_FILE = QORE_SOURCE_FILE

  File class source code
- const Qore::SOURCE_FTPCLIENT = QORE_SOURCE_FTPCLIENT

  FtpClient class source code
- const Qore::SOURCE_HTTPCLIENT = QORE_SOURCE_HTTPCLIENT

  HTTPClient class source code
- const Qore::SOURCE_SOCKET = QORE_SOURCE_SOCKET

  Socket class source code

43.18.1 Detailed Description

43.19 Event Map Constants

Variables

- const Qore::EVENT_MAP

  Maps from Event Constants (the keys) to descriptive strings (the values)
- const Qore::EVENT_SOURCE_MAP

  Maps from Event Source Constants (the keys) to descriptive strings (the values)

43.19.1 Detailed Description

43.20 Event Constants

Variables

- const Qore::EVENT_CHANNEL_CLOSED = QORE_EVENT_CHANNEL_CLOSED

  Raised when a socket or file is closed.
- const Qore::EVENT_CONNECTED = QORE_EVENT_CONNECTED

  Raised when the socket connection has been established.
- const Qore::EVENT_CONNECTING = QORE_EVENT_CONNECTING

  Raised right before a socket connection attempt is made.
- const Qore::EVENT_DATA_READ = QORE_EVENT_DATA_READ

  Raised when data has been read from a file.
- const Qore::EVENT_DATA_WRITTEN = QORE_EVENT_DATA_WRITTEN

  Raised when data has been written to a file.
- const Qore::EVENT_DELETED = QORE_EVENT_DELETED

  Raised when the object being monitored is deleted.
- const Qore::EVENT_FILE_OPENED = QORE_EVENT_FILE_OPENED

  Raised when a file has been successfully opened.
- const Qore::EVENT_FTP_MESSAGE_RECEIVED = QORE_EVENT_FTP_MESSAGE_RECEIVED

  Raised when an FTP reply is received on the control channel.
- const Qore::EVENT_FTP_SEND_MESSAGE = QORE_EVENT_FTP_SEND_MESSAGE

  Raised immediately before an FTP control message is sent.
- const Qore::EVENT_HOSTNAME_LOOKUP = QORE_EVENT_HOSTNAME_LOOKUP

  Raised when a hostname lookup is started.
- const Qore::EVENT_HOSTNAME_RESOLVED = QORE_EVENT_HOSTNAME_RESOLVED

  Raised when a hostname lookup is resolved.
- const Qore::EVENT_HTTP_CHUNKED_DATA_RECEIVED = QORE_EVENT_HTTP_CHUNKED_DATA_↩ RECEIVED

  Raised when a block of HTTP chunked data is received.
- const Qore::EVENT_HTTP_CHUNKED_END = QORE_EVENT_HTTP_CHUNKED_END

  Raised when all HTTP chunked data has been received.
- const Qore::EVENT_HTTP_CHUNKED_START = QORE_EVENT_HTTP_CHUNKED_START

  Raised when HTTP chunked data is about to be received.
- const Qore::EVENT_HTTP_CHUNK_SIZE = QORE_EVENT_HTTP_CHUNK_SIZE

  Raised when the next chunk size for HTTP chunked data is known.
- const Qore::EVENT_HTTP_CONTENT_LENGTH = QORE_EVENT_HTTP_CONTENT_LENGTH

  Raised when the HTTP "`Content-Length`" header is received.
- const Qore::EVENT_HTTP_FOOTERS_RECEIVED = QORE_EVENT_HTTP_FOOTERS_RECEIVED

  Raised when HTTP footers are received.
- const Qore::EVENT_HTTP_MESSAGE_RECEIVED = QORE_EVENT_HTTP_MESSAGE_RECEIVED

  Raised when an HTTP message is received.
- const Qore::EVENT_HTTP_REDIRECT = QORE_EVENT_HTTP_REDIRECT

  Raised when an HTTP redirect message is received.
- const Qore::EVENT_HTTP_SEND_MESSAGE = QORE_EVENT_HTTP_SEND_MESSAGE

  Raised when an HTTP message is sent.
- const Qore::EVENT_OPEN_FILE = QORE_EVENT_OPEN_FILE

  Raised right before a file is opened.
- const Qore::EVENT_PACKET_READ = QORE_EVENT_PACKET_READ

  Raised when a network packet is received.
- const Qore::EVENT_PACKET_SENT = QORE_EVENT_PACKET_SENT

  Raised when a network packet is sent.

- const Qore::EVENT_SSL_ESTABLISHED = QORE_EVENT_SSL_ESTABLISHED

  Raised when SSL communication has been negotiated and established.
- const Qore::EVENT_START_SSL = QORE_EVENT_START_SSL

  Raised when socket SSL negotiation starts.

43.20.1 Detailed Description

43.21 I/O Constants

Variables

- const Qore::stderr = qore(QC_FILE->execSystemConstructor(2))

  system constant for stderr (file descriptor 2)
- const Qore::stdin = qore(QC_FILE->execSystemConstructor(0))

  system constant for stdin (file descriptor 0)
- const Qore::stdout = qore(QC_FILE->execSystemConstructor(1))

  system constant for stdout (file descriptor 1)

43.21.1 Detailed Description

These constants are all instantiations of the File class

None of these constants are available if the parse option PO_NO_TERMINAL_IO is set

43.22 RangeIterator helper functions

Functions

- RangeIterator Qore::xrange (int start, int stop, int step=1)

  Returns a RangeIterator containing an arithmetic progression of integers.
- RangeIterator Qore::xrange (int stop)

  Returns a RangeIterator containing an arithmetic progression of integers with start = 0 and step = 1.

43.22.1 Detailed Description

RangeIterator helper functions allow to use RangeItartor in easier usage in loop statements.

Example

```
foreach my int $i in(xrange(5))
    printf("i=%d\n", $i);
# resulting in:
# i=0
# i=1
# i=2
# i=3
# i=4
# i=5
```

43.22.2 Function Documentation

43.22.2.1 RangeIterator Qore::xrange (int *start*, int *stop*, int *step* = 1)

Returns a RangeIterator containing an arithmetic progression of integers.

Code Flags:

RET_VALUE_ONLY

Example:

```
xrange(2, 5); # 2, 3, 4, 5
xrange(2, -2); # 2, 1, 0, -1, -2
xrange(1, 10, 5); # 1, 6
xrange(0, 10, 5); # 0, 5, 10
xrange(-10, 10, 5); # -10, -5, 0, 5, 10
xrange(10, -10, 5); # 10, 5, 0, -5, -10
```

Parameters

start	the initial value
stop	the final value
step	the step; the default is 1; must be greater than 0; the function throws a `RANGE-ERROR` exception when this argument is < 1

Return values

Returns	a RangeIterator containing an arithmetic progression of integers.

Exceptions

RANGEITERATOR-ERR↩ OR	this exception is thrown if `step` < 1

See also

range

Note

the main difference between range() and xrange() is that range returns a real list and xrange() returns a RangeIterator

Since

Qore 0.8.6

43.22.2.2 RangeIterator Qore::xrange (int *stop*)

Returns a RangeIterator containing an arithmetic progression of integers with start = 0 and step = 1.

This is an overloaded version of xrange(int, int, int) meaning `xrange`(0, `stop`, 1)

Code Flags:

CONSTANT

Example:

```
xrange(1); # 0, 1
xrange(-3); # 0, -1, -2, -3
```

Parameters

stop	the final value

Return values

Returns	a RangeIterator containing an arithmetic progression of integers with start = 0 and step = 1.

See also

range

Note

the main difference between range() and xrange() is that range() returns a real list and xrange() returns a RangeIterator

Since

Qore 0.8.6

43.23 File Stat Constants

Variables

- const Qore::S_IFBLK = S_IFBLK

  Bitmask signifying if the file is a block special (device) file.
- const Qore::S_IFCHR = S_IFCHR

  Bit signifying if the file is a character special (device) file.
- const Qore::S_IFDIR = S_IFDIR

  Bit signifying if the entry is a directory.
- const Qore::S_IFLNK = S_IFLNK

  Bitmask signifying if the file is a symbolic link; equal to 0 on native Windows ports.
- const Qore::S_IFMT = S_IFMT

  File type bitmask
- const Qore::S_IFREG = S_IFREG

  Bit signifying if the file is a regular file.
- const Qore::S_IFSOCK = S_IFSOCK

  Bitmask signifying if the file is a socket file; equal to 0 on native Windows ports.
- const Qore::S_IFWHT = S_IFWHT

  Bitmask signifying if the file is a whiteout file; equal to 0 on native Windows ports.
- const Qore::S_IRGRP = S_IRGRP

  Bit signifying if the file's group has read permissions; equal to 0 on native Windows ports.
- const Qore::S_IROTH = S_IROTH

  Bit signifying if other has read permissions; equal to 0 on native Windows ports.
- const Qore::S_IRUSR = S_IRUSR

  Bit signifying if the file's owner has read permissions.
- const Qore::S_IRWXG = S_IRWXG

  Bitmask giving the RWX mask for the group; equal to 0 on native Windows ports.
- const Qore::S_IRWXO = S_IRWXO

  Bitmask giving the RWX mask for other; equal to 0 on native Windows ports.
- const Qore::S_IRWXU = S_IRWXU

  Bitmask giving the RWX mask for the owner.
- const Qore::S_ISGID = S_ISGID

  Bit signifying set group id on execution; equal to 0 on native Windows ports.
- const Qore::S_ISUID = S_ISUID

  Bit signifying set user id on execution; equal to 0 on native Windows ports.
- const Qore::S_ISVTX = S_ISVTX

  Bit signifying restricted deletes for directories; equal to 0 on native Windows ports.
- const Qore::S_IWGRP = S_IWGRP

  Bit signifying if the file's group has write permissions; equal to 0 on native Windows ports.
- const Qore::S_IWOTH = S_IWOTH

  Bit signifying if other has write permissions; equal to 0 on native Windows ports.
- const Qore::S_IWUSR = S_IWUSR

  Bit signifying if the file's owner has write permissions.
- const Qore::S_IXGRP = S_IXGRP

  Bit signifying if the file's group has execute permissions; equal to 0 on native Windows ports.
- const Qore::S_IXOTH = S_IXOTH

  Bit signifying if other has execute permissions; equal to 0 on native Windows ports.
- const Qore::S_IXUSR = S_IXUSR

  Bit signifying if the file's owner has execute permissions.

43.23.1 Detailed Description

These are values that can be and'ed with the "mode" element of a file's status as returned by Qore::ReadOnlyFile::hstat(), Qore::hstat(), etc, or with element 2 of the status list as returned from Qore::ReadOnlyFile::stat(), Qore::stat(), etc.

43.24 X.509 Verification Constants

Variables

- const Qore::X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH = "X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH"

  Issuer name and serial number of candidate certificate do not match the authority key identifier of the current certificate.
- const Qore::X509_V_ERR_AKID_SKID_MISMATCH = "X509_V_ERR_AKID_SKID_MISMATCH"

  The current candidate issuer certificate was rejected because its subject key identifier was present and did not match the authority key identifier of the current certificate.
- const Qore::X509_V_ERR_APPLICATION_VERIFICATION = "X509_V_ERR_APPLICATION_VERIFICATION"

  Verification failure.
- const Qore::X509_V_ERR_CERT_CHAIN_TOO_LONG = "X509_V_ERR_CERT_CHAIN_TOO_LONG"

  Certificate chain too long.
- const Qore::X509_V_ERR_CERT_HAS_EXPIRED = "X509_V_ERR_CERT_HAS_EXPIRED"

  Certificate has expired.
- const Qore::X509_V_ERR_CERT_NOT_YET_VALID = "X509_V_ERR_CERT_NOT_YET_VALID"

  Certificate is not yet valid.
- const Qore::X509_V_ERR_CERT_REJECTED = "X509_V_ERR_CERT_REJECTED"

  Root CA is marked to reject the specified purpose.
- const Qore::X509_V_ERR_CERT_REVOKED = "X509_V_ERR_CERT_REVOKED"

  Certificate has been revoked.
- const Qore::X509_V_ERR_CERT_SIGNATURE_FAILURE = "X509_V_ERR_CERT_SIGNATURE_FAILURE"

  Certificate signature failure; the signature of the certificate is invalid.
- const Qore::X509_V_ERR_CERT_UNTRUSTED = "X509_V_ERR_CERT_UNTRUSTED"

  Root CA is not marked as trusted for the specified purpose.
- const Qore::X509_V_ERR_CRL_HAS_EXPIRED = "X509_V_ERR_CRL_HAS_EXPIRED"

  CRL has expired.
- const Qore::X509_V_ERR_CRL_NOT_YET_VALID = "X509_V_ERR_CRL_NOT_YET_VALID"

  CRL is not yet valid.
- const Qore::X509_V_ERR_CRL_SIGNATURE_FAILURE = "X509_V_ERR_CRL_SIGNATURE_FAILURE"

  CRL signature failure; the signature of the certificate is invalid.
- const Qore::X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT = "X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT"

  Certificate is self-signed and cannot be found in the trusted list.
- const Qore::X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD = "X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD"

  Format error in certificate's notAfter field (invalid time)
- const Qore::X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD = "X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD"

  Format error in certificate's notBefore field (invalid time)
- const Qore::X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD = "X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD"

  Format error in CRL's lastUpdate field (invalid time)
- const Qore::X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD = "X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD"

  Format error in CRL's nextUpdate field (invalid time)
- const Qore::X509_V_ERR_INVALID_CA = "X509_V_ERR_INVALID_CA"

  Invalid CA certificate.

- const Qore::X509_V_ERR_INVALID_PURPOSE = "X509_V_ERR_INVALID_PURPOSE"

  The certificate cannot be used for the specified purpose.
- const Qore::X509_V_ERR_KEYUSAGE_NO_CERTSIGN = "X509_V_ERR_KEYUSAGE_NO_CERTSIGN"

  The keyUsage extension does not permit certificate signing.
- const Qore::X509_V_ERR_OUT_OF_MEM = "X509_V_ERR_OUT_OF_MEM"

  Out of memory error.
- const Qore::X509_V_ERR_PATH_LENGTH_EXCEEDED = "X509_V_ERR_PATH_LENGTH_EXCEEDED"

  The basicConstraints pathlength parameter has been exceeded.
- const Qore::X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN = "X509_V_ERR_SELF_SIGNED_CERT_↩IN_CHAIN"

  Self signed certificate in certificate chain.
- const Qore::X509_V_ERR_SUBJECT_ISSUER_MISMATCH = "X509_V_ERR_SUBJECT_ISSUER_MIS↩MATCH"

  The current candidate issuer certificate was rejected because its subject name did not match the issuer name of the current certificate.
- const Qore::X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY = "X509_V_ERR_UNABLE_↩TO_DECODE_ISSUER_PUBLIC_KEY"

  Unable to decode issuer public key (SubjectPublicKeyInfo)
- const Qore::X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE = "X509_V_ERR_UNABLE_T↩O_DECRYPT_CERT_SIGNATURE"

  Unable to decrypt certificate's signature. This means that the actual signature value could not be determined rather than it not matching the expected value; this is only meaningful for RSA.
- const Qore::X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE = "X509_V_ERR_UNABLE_TO↩_DECRYPT_CRL_SIGNATURE"

  Unable to decrypt CRL's signature.
- const Qore::X509_V_ERR_UNABLE_TO_GET_CRL = "X509_V_ERR_UNABLE_TO_GET_CRL"

  Unable to get certificate CRL.
- const Qore::X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT = "X509_V_ERR_UNABLE_TO_GET_IS↩SUER_CERT"

  Unable to get issuer certificate.
- const Qore::X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY = "X509_V_ERR_UNABLE_T↩O_GET_ISSUER_CERT_LOCALLY"

  Unable to get local issuer certificate. This normally means the list of trusted certificates is not complete.
- const Qore::X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE = "X509_V_ERR_UNABLE_TO_↩VERIFY_LEAF_SIGNATURE"

  Unable to verify the first certificate.
- const Qore::X509_V_OK = "X509_V_OK"

  Verification OK.
- const Qore::X509_VerificationReasons

  maps from varification strings to verification code descriptions

43.24.1 Detailed Description

These are string contants for values returned by the following methods:

- FtpClient::verifyPeerCertificate()
- HTTPClient::verifyPeerCertificate()
- Socket::verifyPeerCertificate()

43.25 Network Address Family Constants

Variables

- const Qore::AFMap = qore(get_network_address_family_map())

  mapping from Network Address Family Constants to string codes
- const Qore::AFStrMap

  mapping from network address family string codes to Network Address Family Constants
- const Qore::AF_INET = AF_INET

  IPv4 address family.
- const Qore::AF_INET6 = AF_INET6

  IPv6 address family.
- const Qore::AF_LOCAL = AF_LOCAL

  POSIX synonym for AF_UNIX.
- const Qore::AF_UNIX = AF_UNIX

  UNIX domain address family (UNIX socket files)
- const Qore::AF_UNSPEC = AF_UNSPEC

  unspecified address family

43.25.1 Detailed Description

These are the possible network address family constants

43.26 Network Address Information Constants

Variables

- const Qore::AI_ADDRCONFIG = AI_ADDRCONFIG

  if this bit is set, addresses of each family are returned only if they are configured on the system
- const Qore::AI_ALL = AI_ALL

  If this bit is set along with AI_V4MAPPED then all matching IPv6 and IPv4 addresses are returned.
- const Qore::AI_CANONNAME = AI_CANONNAME

  If this bit is set, then getaddrinfo() will return the canonical name of the hostname in the "`canonname`" key of the first element returned.
- const Qore::AI_NUMERICHOST = AI_NUMERICHOST

  If this bit is set, then the host is assumed to be an address and no hostname lookup will be preformed.
- const Qore::AI_NUMERICSERV = AI_NUMERICSERV

  If this bit is set, then the service is assumed to be a numeric port string, and no service lookup will be performed.
- const Qore::AI_PASSIVE = AI_PASSIVE

  If this bit is set, then the returned information should be usable for a call to Socket::bind()
- const Qore::AI_V4MAPPED = AI_V4MAPPED

  If this bit is set, getaddrinfo() will return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses/.

43.26.1 Detailed Description

These are the possible network address information constants for the getaddrinfo() function; if any of these constants are not defined on the current platform, they will be assigned to 0 in Qore

43.26.2 Variable Documentation

43.26.2.1 const Qore::AI_NUMERICSERV = AI_NUMERICSERV

If this bit is set, then the service is assumed to be a numeric port string, and no service lookup will be performed.

If the AI_NUMERICSERV bit is set, then any service name string supplied will be treated as a numeric port string. Otherwise, a `QOREADDRINFO-GETINFO-ERROR` exception will be thrown due to the `EAI_NONAME` error raised internally. This bit prevents any type of name resolution service (for example, NIS+) from being invoked (this constant is operating system dependent; it is not available on all operating systems; for example, it is not available on Solaris 8).

43.26.2.2 const Qore::AI_PASSIVE = AI_PASSIVE

If this bit is set, then the returned information should be usable for a call to Socket::bind()

In this case, if the hostname is not given, then the return value will be usable for binding on all addresses for the given family.

If the AI_PASSIVE bit is not set, the returned socket address structure will be ready for use in a call to Socket::connect().

The IP address portion of the socket address structure will be set to the loopback address if hostname is NOTHING and AI_PASSIVE is not set.

43.26.2.3 const Qore::AI_V4MAPPED = AI_V4MAPPED

If this bit is set, getaddrinfo() will return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses/.

The AI_V4MAPPED flag is ignored unless family is AF_INET6

43.27 Network Protocol Constants

Variables

- const Qore::IPPROTO_TCP = IPPROTO_TCP

  for the TCP protocol
- const Qore::IPPROTO_UDP = IPPROTO_UDP

  for the UDP protocol

43.27.1 Detailed Description

These are the possible network protocol constants

43.28 Socket Type Constants

Variables

- const Qore::SOCK_DGRAM = SOCK_DGRAM

  for datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length
- const Qore::SOCK_RAW = SOCK_RAW

  raw socket interface, only available to the superuser, untested
- const Qore::SOCK_STREAM = SOCK_STREAM

  for sequenced, reliable, two-way connection-based byte streams (the default)

43.28.1 Detailed Description

These are the possible socket type constants

43.29 Terminal Attribute Local Mode Constants

Variables

- const Qore::ALTWERASE = ALTWERASE

  use alternate WERASE algorithm
- const Qore::ECHO = ECHO

  enable echoing
- const Qore::ECHOCTL = ECHOCTL

  echo control chars as ^(Char)
- const Qore::ECHOE = ECHOE

  visually erase chars
- const Qore::ECHOKE = ECHOKE

  visual erase for line kill
- const Qore::ECHONL = ECHONL

  echo NL even if ECHO is off
- const Qore::ECHOPRT = ECHOPRT

  visual erase mode for hardcopy
- const Qore::EXTPROC = EXTPROC

  external processing
- const Qore::FLUSHO = FLUSHO

  output being flushed (state)
- const Qore::ICANON = ICANON

  canonicalize input lines
- const Qore::IEXTEN = IEXTEN

  enable DISCARD and LNEXT
- const Qore::ISIG = ISIG

  enable signals INTR, QUIT, [D]SUSP
- const Qore::NOFLSH = NOFLSH

  don't flush after interrupt
- const Qore::NOKERNINFO = NOKERNINFO

  no kernel output from VSTATUS
- const Qore::PENDIN = PENDIN

  retype pending input (state)
- const Qore::TOSTOP = TOSTOP

  stop background jobs from output

43.29.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.30 Terminal Attribute Control Mode Constants

Variables

- const Qore::CCAR_OFLOW = CCAR_OFLOW

  DCD flow control of output.
- const Qore::CCTS_OFLOW = CCTS_OFLOW

  CTS flow control of output.
- const Qore::CDSR_OFLOW = CDSR_OFLOW

  DSR flow control of output.
- const Qore::CLOCAL = CLOCAL

  ignore modem status lines
- const Qore::CREAD = CREAD

  enable receiver
- const Qore::CRTSCTS = CRTSCTS

  CTS flow control of output and RTS flow control of input.
- const Qore::CRTS_IFLOW = CRTS_IFLOW

  RTS flow control of input.
- const Qore::CS5 = CS5

  character size mask: 5 bits
- const Qore::CS6 = CS6

  character size mask: 6 bits
- const Qore::CS7 = CS7

  character size mask: 7 bits
- const Qore::CS8 = CS8

  character size mask: 8 bits
- const Qore::CSIZE = CSIZE

  character size mask
- const Qore::CSTOPB = CSTOPB

  send 2 stop bits
- const Qore::HUPCL = HUPCL

  hang up on last close
- const Qore::MDMBUF = MDMBUF

  old name for CCAR_OFLOW
- const Qore::PARENB = PARENB

  parity enable
- const Qore::PARODD = PARODD

  odd parity, else even

43.30.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.31 Terminal Attributes Output Mode Constants

Variables

- const Qore::OCRNL = OCRNL

  map CR to NL on output
- const Qore::ONLCR = ONLCR

  map NL to CR-NL (ala CRMOD)
- const Qore::ONLRET = ONLRET

  NL performs CR function.
- const Qore::ONOCR = ONOCR

  no CR output at column 0
- const Qore::ONOEOT = ONOEOT

  discard EOT's (^D) on output
- const Qore::OPOST = OPOST

  enable following output processing
- const Qore::OXTABS = OXTABS

  expand tabs to spaces

43.31.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.32 Terminal Attributes Input Mode Constants

Variables

- const Qore::BRKINT = BRKINT
  map BREAK to SIGINTR
- const Qore::ICRNL = ICRNL
  map CR to NL (ala CRMOD)
- const Qore::IGNBRK = IGNBRK
  ignore BREAK condition
- const Qore::IGNCR = IGNCR
  ignore CR
- const Qore::IGNPAR = IGNPAR
  ignore (discard) parity errors
- const Qore::IMAXBEL = IMAXBEL
  ring bell on input queue full
- const Qore::INLCR = INLCR
  map NL into CR
- const Qore::INPCK = INPCK
  enable checking of parity errors
- const Qore::ISTRIP = ISTRIP
  strip 8th bit off chars
- const Qore::IXANY = IXANY
  any char will restart after stop
- const Qore::IXOFF = IXOFF
  enable input flow control
- const Qore::IXON = IXON
  enable output flow control
- const Qore::PARMRK = PARMRK
  mark parity and framing errors

43.32.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.33 Terminal Attributes Control Character Constants

Variables

- const Qore::VDISCARD = VDISCARD

  subscript for the VDISCARD character
- const Qore::VDSUSP = VDSUSP

  subscript for the VDSUSP character
- const Qore::VEOF = VEOF

  subscript for the EOF character
- const Qore::VEOL = VEOL

  subscript for the EOL character
- const Qore::VEOL2 = VEOL2

  subscript for the EOL2 character
- const Qore::VERASE = VERASE

  subscript for the VERASE character
- const Qore::VINTR = VINTR

  subscript for the VINTR character
- const Qore::VKILL = VKILL

  subscript for the VKILL character
- const Qore::VLNEXT = VLNEXT

  subscript for the VLNEXT character
- const Qore::VMIN = VMIN

  subscript for the VMIN value
- const Qore::VQUIT = VQUIT

  subscript for the VQUIT character
- const Qore::VREPRINT = VREPRINT

  subscript for the VREPRINT character
- const Qore::VSTART = VSTART

  subscript for the VSTART character
- const Qore::VSTATUS = VSTATUS

  subscript for the character
- const Qore::VSTOP = VSTOP

  subscript for the VSTOP character
- const Qore::VSUSP = VSUSP

  subscript for the VSUSP character
- const Qore::VTIME = VTIME

  subscript for the VTIME value
- const Qore::VWERASE = VWERASE

  subscript for the VWERASE character
- const Qore::_POSIX_VDISABLE = _POSIX_VDISABLE

  if the value of any key is this value, it means that the key is disabled

43.33.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.34 Terminal Attributes Terminal Setting Constants

Variables

- const Qore::TCSADRAIN = TCSADRAIN

  drain output, then change
- const Qore::TCSAFLUSH = TCSAFLUSH

  drain output, flush input
- const Qore::TCSANOW = TCSANOW

  make change immediate
- const Qore::TCSASOFT = TCSASOFT

  flag - don't alter hardware state

43.34.1 Detailed Description

If any of the constants in this group is not defined on the host system, then it will be assigned to 0 in Qore.

43.35 Compression Functions

Functions

- binary Qore::bunzip2_to_binary (binary bin)

  Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a binary object.
- nothing Qore::bunzip2_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::bunzip2_to_string (binary bin, __7_ string encoding)

  Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a string.
- nothing Qore::bunzip2_to_string ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::bzip2 (binary bin, softint level=BZ2_DEFAULT_COMPRESSION)

  Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.
- binary Qore::bzip2 (string str, softint level=BZ2_DEFAULT_COMPRESSION)

  Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.
- nothing Qore::bzip2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::compress (string str, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.
- binary Qore::compress (binary bin, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.
- nothing Qore::compress ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::gunzip_to_binary (binary bin)

  Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a binary object of the uncompressed data.
- nothing Qore::gunzip_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::gunzip_to_string (binary bin, __7_ string encoding)

  Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a string of the uncompressed datas.
- nothing Qore::gunzip_to_string ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::gzip (string str, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.
- binary Qore::gzip (binary bin, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.
- nothing Qore::gzip ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::uncompress_to_binary (binary bin)

  Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a binary object of the decompressed data.
- nothing Qore::uncompress_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::uncompress_to_string (binary bin, __7_ string encoding)

  Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a string of the decompressed data.
- nothing Qore::uncompress_to_string ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.35.1 Detailed Description

These functions work with compression and decompression

43.35.2 Function Documentation

43.35.2.1 binary Qore::bunzip2_to_binary (binary *bin*)

Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a binary object.

Parameters

bin	the compressed data to decompress

Returns

the uncompressed data as a binary object

Example:

```
my string $str = bunzip2_to_binary($bzip2_string);
```

Exceptions

BZIP2-DECOMPRESS-ERROR	the bzip2 library returned an internal error during processing (possibly due to corrupt input data)

43.35.2.2 nothing Qore::bunzip2_to_binary ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.3 string Qore::bunzip2_to_string (binary *bin*, __7_ string *encoding*)

Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a string.

Parameters

bin	the compressed data to decompress

encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed.

Returns

the uncompressed data as a string

Example:

```
my string $str = bunzip2_to_string($bzip2_string, "iso-8859-1");
```

Exceptions

BZIP2-DECOMPRESS-ERROR	the bzip2 library returned an internal error during processing (possibly due to corrupt input data)

43.35.2.4 nothing Qore::bunzip2_to_string ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.5 binary Qore::bzip2 (binary *bin*, softint *level* = BZ2_DEFAULT_COMPRESSION)

Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.

Parameters

bin	the data to compress
level	the compression level, must be a value between 1 and 9 inclusive, 1 = the least compression (and taking the least memory), 9 = the most compression (using the most memory). An invalid option passed to this argument will result in a BZLIB2-LEVEL-ERROR exception being raised.

Returns

the compressed data as a binary object

Example:

```
my binary $bin = compress($data);
```

Exceptions

BZLIB2-LEVEL-ERROR	level must be between 1 - 9
BZIP2-COMPRESS-ERROR	the bzip2 library returned an error during processing

43.35.2.6 binary Qore::bzip2 (string *str*, softint *level* = BZ2_DEFAULT_COMPRESSION)

Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.

Strings are compressed without the trailing null character

Parameters

str	the data to compress
level	the compression level, must be a value between 1 and 9 inclusive, 1 = the least compression (and taking the least memory), 9 = the most compression (using the most memory). An invalid option passed to this argument will result in a BZLIB2-LEVEL-ERROR exception being raised.

Returns

the compressed data as a binary object

Example:

```
my binary $bin = compress($str);
```

Exceptions

BZLIB2-LEVEL-ERROR	level must be between 1 - 9
BZIP2-COMPRESS-ERROR	the bzip2 library returned an error during processing

43.35.2.7 nothing Qore::bzip2 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.8 binary Qore::compress (string *str*, int *level* = Z_DEFAULT_COMPRESSION)

Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.

Note that strings are compressed without the trailing null character.

Parameters

str	The string to compress
level	Specifies the compression level; must be an integer between 1 and 9, 9 meaning the highest compression level. The default value Z_DEFAULT_COMPRESSION gives a tradeoff between speed and compression size

Returns

a binary object of the compressed data

Example:

```
my binary $bin = compress($str_data);
```

Exceptions

ZLIB-LEVEL-ERROR	level must be between 1 - 9 or -1
ZLIB-ERROR	zlib returned an error while processing

43.35.2.9 binary Qore::compress (binary *bin*, int *level* = Z_DEFAULT_COMPRESSION)

Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.

Note that strings are compressed without the trailing null character.

Parameters

bin	The binary object to compress
level	Specifies the compression level; must be an integer between 1 and 9, 9 meaning the highest compression level. The default value Z_DEFAULT_COMPRESSION gives a tradeoff between speed and compression size

Returns

a binary object of the compressed data

Example:

```
my binary $bin = compress($bin_data);
```

Exceptions

ZLIB-LEVEL-ERROR	level must be between 1 - 9 or -1
ZLIB-ERROR	zlib returned an error while processing

43.35.2.10 nothing Qore::compress ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.11 binary Qore::gunzip_to_binary (binary *bin*)

Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a binary object of the uncompressed data.

Parameters

bin	the compressed data to decompress

Returns

the uncompressed data as a binary object

Example:

```
my binary $bin = gunzip_to_string($data);
```

Exceptions

ZLIB-ERROR	The zlib library returned an error during processing (possibly due to corrupt input data)

43.35.2.12 nothing Qore::gunzip_to_binary ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.13 string Qore::gunzip_to_string (binary *bin*, __7_ string *encoding*)

Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a string of the uncompressed datas.

Parameters

bin	the compressed data to decompress
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed.

Returns

the uncompressed data as a string

Example:

```
my string $str = gunzip_to_string($bin, "iso-8859-1");
```

Exceptions

ZLIB-ERROR	The zlib library returned an error during processing (possibly due to corrupt input data)

43.35.2.14 nothing Qore::gunzip_to_string ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.15 binary Qore::gzip (string *str*, int *level* = Z_DEFAULT_COMPRESSION)

Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.

Strings are compressed without the trailing null character

Parameters

str	the data to compress
level	the compression level, must be a value between 1 and 9 inclusive, 1 = the least compression (and taking the least memory), 9 = the most compression (using the most memory). An invalid option passed to this argument will result in a ZLIB-LEVEL-ERROR exception being raised.

Returns

the compressed data as a binary object

Example:

```
my binary $data = bzip($str);
```

Exceptions

ZLIB-LEVEL-ERROR	level must be between 1 - 9 or -1
ZLIB-ERROR	The zlib library returned an error during processing (should not normally happen during compression)

43.35.2.16 binary Qore::gzip (binary *bin*, int *level* = Z_DEFAULT_COMPRESSION)

Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.

Parameters

bin	the data to compress
level	the compression level, must be a value between 1 and 9 inclusive, 1 = the least compression (and taking the least memory), 9 = the most compression (using the most memory). An invalid option passed to this argument will result in a ZLIB-LEVEL-ERROR exception being raised.

Returns

the compressed data as a binary object

Example:

```
my binary $data = gzip($bin);
```

Exceptions

ZLIB-LEVEL-ERROR	level must be between 1 - 9 or -1
ZLIB-ERROR	The zlib library returned an error during processing (should not normally happen during compression)

43.35.2.17 nothing Qore::gzip ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.18 binary Qore::uncompress_to_binary (binary *bin*)

Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a binary object of the decompressed data.

Parameters

bin	the data to decompress

Returns

a binary object of the decompressed data

Example:

```
my binary $bin = uncompress_to_binary($data);
```

Exceptions

ZLIB-ERROR	The zlib library returned an error during processing (possibly due to corrupt input data)

43.35.2.19 nothing Qore::uncompress_to_binary ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.35.2.20 string Qore::uncompress_to_string (binary *bin*, __7_ string *encoding*)

Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a string of the decompressed data.

Parameters

bin	the compressed data to decompress
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed.

Returns

the uncompressed data as a string

Example:

```
my string $str = uncompress_to_string($bin, "iso-8859-1");
```

Exceptions

ZLIB-ERROR	The zlib library returned an error during processing (possibly due to corrupt input data)

43.35.2.21 nothing Qore::uncompress_to_string ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.36 Compression Constants

Variables

- const Qore::BZ2_DEFAULT_COMPRESSION = BZ2_DEFAULT_COMPRESSION

  gives the default compression level for the bzip2() function, providing maximum compression (value: 9)
- const Qore::Z_DEFAULT_COMPRESSION = Z_DEFAULT_COMPRESSION

  gives the default compression level for the compress() and gzip() functions, providing a tradeoff between compression speed and compression size

43.36.1 Detailed Description

43.37 Context Functions

Functions

- bool Qore::cx_first ()

  Returns True if currently iterating the first element in a context statement, False if not.
- bool Qore::cx_last ()

  Returns True if currently iterating the last element in a context statement, False if not.
- int Qore::cx_pos ()

  Returns the current row number within the active context statement (starting from 0)
- int Qore::cx_total ()

  Returns the total number of rows within the active context statement.
- any Qore::cx_value (string key)

  Returns the current value of the given column while iterating a context statement.

43.37.1 Detailed Description

43.37.2 Function Documentation

43.37.2.1 bool Qore::cx_first ( )

Returns True if currently iterating the first element in a context statement, False if not.

Returns

True if currently iterating the first element in a context statement, False if not

Code Flags:

RET_VALUE_ONLY

Example:

```
context ($ds.select($sql)) {
    if (cx_first())
        print("first row!\n");
}
```

Exceptions

CONTEXT-ERROR	this exception is thrown if called without an active context statement

Since

Qore 0.8.4

43.37.2.2 bool Qore::cx_last ( )

Returns True if currently iterating the last element in a context statement, False if not.

Returns

True if currently iterating the last element in a context statement, False if not

Code Flags:

RET_VALUE_ONLY

Example:

```
context ($ds.select($sql)) {
    if (cx_last())
        print("last row!\n");
}
```

Exceptions

CONTEXT-ERROR	this exception is thrown if called without an active context statement

Since

Qore 0.8.4

43.37.2.3 int Qore::cx_pos ( )

Returns the current row number within the active context statement (starting from 0)

Returns

the current row number within the active context statement (starting from 0)

Code Flags:

RET_VALUE_ONLY

Example:

```
context ($ds.select($sql)) {
    printf("row %d/%d:\n", cx_pos(), cx_total());
}
```

Exceptions

CONTEXT-ERROR	this exception is thrown if called without an active context statement

Since

Qore 0.8.4

43.37.2.4 int Qore::cx_total ( )

Returns the total number of rows within the active context statement.

Returns

the total number of rows within the active context statement

Code Flags:

RET_VALUE_ONLY

Example:

```
context ($ds.select($sql)) {
    printf("row %d/%d:\n", cx_total(), cx_total());
}
```

Exceptions

CONTEXT-ERROR	this exception is thrown if called without an active context statement

Since

Qore 0.8.4

43.37.2.5 any Qore::cx_value (string *key*)

Returns the current value of the given column while iterating a context statement.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name to return the value for

Returns

the current value of the given column while iterating a context statement

Example:

```
context ($ds.select($sql)) {
    my any $val = cx_value("column1");
}
```

Exceptions

CONTEXT-ERROR	this exception is thrown if called without an active context statement
ENCODING-CONVERSI↩ ON-ERROR	the given string could not be converted to the default character encoding

Since

Qore 0.8.4

43.38 Crytographic Functions

Functions

- binary Qore::blowfish_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for the* `blowfish algorithm` *and returns a binary object of the decrypted data.*
- string Qore::blowfish_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for the* `blowfish algorithm` *and returns a string of the decrypted data.*
- binary Qore::blowfish_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for the* `blowfish algorithm`
- binary Qore::cast5_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for the* `CAST5 algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- string Qore::cast5_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for the* `CAST5 algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::cast5_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for the* `CAST5 algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::des_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for the* `DES algorithm` *using an 8-byte key.*
- string Qore::des_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for the* `DES algorithm` *using an 8-byte key.*
- binary Qore::des_ede3_decrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for the three-key triple* `DES algorithm` *using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.*
- string Qore::des_ede3_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for the three-key triple* `DES algorithm` *using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.*
- binary Qore::des_ede3_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for the three-key triple* `DES algorithm` *using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.*
- binary Qore::des_ede_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)*
- string Qore::des_ede_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)*
- binary Qore::des_ede_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)*
- binary Qore::des_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for the* `DES algorithm` *using an 8-byte key.*
- binary Qore::des_random_key ()

  Returns a binary object of a random key for the `DES algorithm`
- binary Qore::desx_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `DESX algorithm` *using a 24-byte key and an optional 8-byte initialization vector.*

- string Qore::desx_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `DESX algorithm` *using a 24-byte key and an optional 8-byte initialization vector.*
- binary Qore::desx_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for RSA's* `DESX algorithm` *using a 24-byte key and an optional 8-byte initialization vector.*
- binary Qore::rc2_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- string Qore::rc2_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::rc2_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::rc4_decrypt (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Alleged `RC4 cipher algorithm`*, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.*
- string Qore::rc4_decrypt_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Alleged `RC4 cipher algorithm`*, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::rc4_encrypt (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Alleged `RC4 cipher algorithm`*, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::rc5_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- string Qore::rc5_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*
- binary Qore::rc5_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the `Cipher Block Chaining` *function for RSA's* `RC2(tm) algorithm` *using a variable-length key and an optional 8-byte initialization vector.*

43.38.1 Detailed Description

Qore's cryptography support is provided by the OpenSSL library. Each of the encryption and decryption functions in this section accept an optional initialization vector, which is data used as initial input for the first block in chained encryption algorithms. Subsequent blocks take input from the last block encrypted/decrypted. If an initialization vector is not supplied, a default value of 8 zero bytes will be used (see Qore::DefaultIV).

See also:

- Digest (Hash) Functions
- HMAC Functions
- Crytographic Contants

Some functions require fixed-length keys, and some allow the use of variable-length keys. For functions requiring fixed-length keys any excess bytes are ignored. The same applies to initialization vector arguments.

The following is an example of a function that uses `/dev/random` on UNIX to read in a random key for use with encryption functions:

```
# read a key from /dev/random and return the key
binary sub get_key(int $size) {
    # throw an exception if an invalid key size was passed
    if (!$size || $size < 0)
        throw "GET-KEY-ERROR", sprintf("invalid size = %n", $size);
    my File $f();
    # File::open2() will throw an exception if /dev/random cannot be opened for reading
    $f.open2("/dev/random");
    return $f.readBinary($size);
}
```

43.38.2 Function Documentation

43.38.2.1 binary Qore::blowfish_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for the blowfish algorithm and returns a binary object of the decrypted data.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-length key (recommended 16 bytes or more)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = blowfish_decrypt_cbc($data, $key);
```

Exceptions

BLOWFISH-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.2 string Qore::blowfish_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, __7_ string *encoding*)

Decrypts data using the Cipher Block Chaining function for the blowfish algorithm and returns a string of the decrypted data.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted

key	a variable-length key (recommended 16 bytes or more)
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $str = blowfish_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

BLOWFISH-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.3 binary Qore::blowfish_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for the blowfish algorithm

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	a variable-length key (recommended 16 bytes or more)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = blowfish_encrypt_cbc($data, $key);
```

Exceptions

BLOWFISH-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.4 binary Qore::cast5_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = cast5_decrypt_cbc($data, $key);
```

Exceptions

CAST5-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.5 string Qore::cast5_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $bin = cast5_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

CAST5-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.6 binary Qore::cast5_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	a variable-width encryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = cast5_encrypt_cbc($data, $key);
```

Exceptions

CAST5-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.7 binary Qore::des_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 8 bytes long (only the first 8 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = des_decrypt_cbc($data, $key);
```

Exceptions

DES-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)

DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.8 string Qore::des_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, __7_ string *encoding*)

Decrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 8 bytes long (only the first 8 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $bin = des_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

DES-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.9 binary Qore::des_ede3_decrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 24 bytes long (only the first 24 bytes will be used for the three 8-byte keys)

iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = des_ede3_decrypt_cbc($data, $key);
```

Exceptions

DES-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.10 string Qore::des_ede3_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 24 bytes long (only the first 24 bytes will be used for the three 8-byte keys)
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $bin = des_ede3_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

DES-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.11 binary Qore::des_ede3_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	the encryption key must be at least 24 bytes long (only the first 24 bytes will be used for the three 8-byte keys)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = des_ede3_encrypt_cbc($data, $key);
```

Exceptions

DES-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.12 binary Qore::des_ede_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 16 bytes long (only the first 16 bytes will be used for the two 8-byte keys)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = des_ede_decrypt_cbc($data, $key);
```

Exceptions

DES-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.13 string Qore::des_ede_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 16 bytes long (only the first 16 bytes will be used for the two 8-byte keys)
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $bin = des_ede_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

DES-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.14 binary Qore::des_ede_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	the encryption key must be at least 16 bytes long (only the first 16 bytes will be used for the two 8-byte keys)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = des_ede_encrypt_cbc($data, $key);
```

Exceptions

DES-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.15 binary Qore::des_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	the encryption key must be at least 8 bytes long (only the first 8 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = des_encrypt_cbc($data, $key);
```

Exceptions

DES-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DES-KEY-ERROR	invalid key (too short)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.16 binary Qore::des_random_key ( )

Returns a binary object of a random key for the DES algorithm

Returns

a binary object of a random key for the DES algorithm

Code Flags:

CONSTANT

Example:

```
my binary $bin = des_random_key();
```

43.38.2.17 binary Qore::desx_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 24 bytes long (only the first 24 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = desx_decrypt_cbc($data, $key);
```

Exceptions

DESX-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DESX-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.18 string Qore::desx_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	the decryption key must be at least 24 bytes long (only the first 24 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a binary object of the decrypted data

Example:

```
my string $str = desx_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

DESX-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DESX-KEY-ERROR	invalid key (too short)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.19 binary Qore::desx_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	the encryption key must be at least 24 bytes long (only the first 24 bytes will be used)
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = desx_encrypt_cbc($data, $key);
```

Exceptions

DESX-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)

DESX-KEY-ERROR	invalid key (too short)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.20 binary Qore::rc2_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = rc2_decrypt_cbc($data, $key);
```

Exceptions

RC2-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.21 string Qore::rc2_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $str = rc2_decrypt_cbc_to_string($data, $key);
```

Exceptions

RC2-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.22 binary Qore::rc2_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	a variable-width encryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = rc2_encrypt_cbc($data, $key);
```

Exceptions

RC2-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.23 binary Qore::rc4_decrypt (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key

iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = rc4_decrypt($data, $key);
```

Exceptions

RC4-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.24 string Qore::rc4_decrypt_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, *__7_* string *encoding*)

Decrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $str = rc4_decrypt_to_string($encrypted_data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

RC4-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.25 binary Qore::rc4_encrypt (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	a variable-width encryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = rc4_encrypt($data, $key);
```

Exceptions

RC4-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.38.2.26 binary Qore::rc5_decrypt_cbc (binary *data*, data *key*, data *iv* = Qore::DefaultIV)

Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Platform Availability:

Qore::Option::HAVE_RC5

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the decrypted data

Example:

```
my binary $bin = rc2_decrypt_cbc($data, $key);
```

Exceptions

RC5-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

43.38.2.27 string Qore::rc5_decrypt_cbc_to_string (binary *data*, data *key*, data *iv* = Qore::DefaultIV, __7_ string *encoding*)

Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Platform Availability:

Qore::Option::HAVE_RC5

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be decrypted
key	a variable-width decryption key
iv	the initialization vector must be at least 8 bytes long if present
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string of the decrypted data

Example:

```
my string $str = rc2_decrypt_cbc_to_string($data, $key,
      NOTHING, "iso-8859-1");
```

Exceptions

RC5-DECRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
DECRYPT-ERROR	processing error decrypting the data (for example invalid data)

Since

Qore 0.8.4 added the encoding parameter to specify the output encoding of the string

43.38.2.28 binary Qore::rc5_encrypt_cbc (data *data*, data *key*, data *iv* = Qore::DefaultIV)

Encrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.

Platform Availability:

Qore::Option::HAVE_RC5

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to be encrypted
key	a variable-width encryption key
iv	the initialization vector must be at least 8 bytes long if present

Returns

a binary object of the encrypted data

Example:

```
my binary $bin = rc2_encrypt_cbc($data, $key);
```

Exceptions

RC5-ENCRYPT-PARAM-ERROR	invalid initialization vector (less than 8 bytes)
ENCRYPT-ERROR	processing error encrypting the data (should not normally happen)

43.39 Digest (Hash) Functions

Functions

- string Qore::DSS (data data)

  Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a hex string.
- string Qore::DSS1 (data data)

  Returns the DSS1 message digest (based on SHA1 and DSA) of the supplied argument as a hex string.
- binary Qore::DSS1_bin (data data)

  Returns the DSS1 message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.
- binary Qore::DSS_bin (data data)

  Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.
- string Qore::MD2 (data data)

  Returns the MD2 message digest of the supplied argument as a hex string.
- binary Qore::MD2_bin (data data)

  Returns the MD2 message digest of the supplied argument as binary object.
- string Qore::MD4 (data data)

  Returns the MD4 message digest of the supplied argument as a hex string.
- binary Qore::MD4_bin (data data)

  Returns the MD4 message digest of the supplied argument as a binary object.
- string Qore::MD5 (data data)

  Returns the MD5 message digest of the supplied argument as a hex string.
- binary Qore::MD5_bin (data data)

  Returns the MD5 message digest of the supplied argument as a binary object.
- string Qore::MDC2 (data data)

  Returns the MDC2 message digest of the supplied argument as a hex string.
- binary Qore::MDC2_bin (data data)

  Returns the MDC2 message digest of the supplied argument as a binary object.
- string Qore::RIPEMD160 (data data)

  Returns the RIPEMD message digest of the supplied argument as a hex string.
- binary Qore::RIPEMD160_binary (data data)

  Returns the RIPEMD message digest of the supplied argument as a binary object.
- string Qore::SHA (data data)

  Returns the SHA (outdated SHA-0) message digest of the supplied argument as a hex string.
- string Qore::SHA1 (data data)

  Returns the SHA1 message digest of the supplied argument as a hex string.
- binary Qore::SHA1_bin (data data)

  Returns the SHA1 message digest of the supplied argument as a binary object.
- string Qore::SHA224 (data data)

  Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary Qore::SHA224_bin (data data)

  Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string Qore::SHA256 (data data)

  Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary Qore::SHA256_bin (data data)

  Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string Qore::SHA384 (data data)

  Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary Qore::SHA384_bin (data data)

  Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string Qore::SHA512 (data data)

Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a hex string.

- binary Qore::SHA512_bin (data data)

  Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a binary object.

- binary Qore::SHA_bin (data data)

  Returns the SHA (outdated SHA-0) message digest of the supplied argument as a binary object.

43.39.1 Detailed Description

Qore's cryptography support is provided by the OpenSSL library.

A cryptographic hash function is a hash function; that is, an algorithm that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value. The data to be encoded are often called the "message," and the hash value is sometimes called the message digest or simply digest.

For more info: Wikipedia's Cryptographic hash function article.

See also:

- Crytographic Functions
- HMAC Functions
- Crytographic Contants

43.39.2 Function Documentation

43.39.2.1 string Qore::DSS (data *data*)

Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d")

Example:

```
my string $str = DSS("hello"); # returns "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

Exceptions

DSS-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

DSS_bin()

Note

this digest algorithm is considered outdated and is included for backwards-compatibility

43.39.2.2 string Qore::DSS1 (data *data*)

Returns the DSS1 message digest (based on SHA1 and DSA) of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d")

Example:

```
my string $str = DSS1("hello"); # returns "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

Exceptions

DSS1-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

DSS1_bin()

Note

this digest algorithm is considered outdated and is included for backwards-compatibility

43.39.2.3 binary Qore::DSS1_bin (data *data*)

Returns the DSS1 message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = DSS1_bin("hello");
```

Exceptions

DSS1-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

DSS1()

Note

this digest algorithm is considered outdated and is included for backwards-compatibility

43.39.2.4 binary Qore::DSS_bin (data *data*)

Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = DSS_bin("hello");
```

Exceptions

DSS-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

DSS()

Note

this digest algorithm is considered outdated and is included for backwards-compatibility

43.39.2.5 string Qore::MD2 (data *data*)

Returns the MD2 message digest of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_MD2

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "a9046c73e00331af68917d3804f70655")

Example:

```
my string $str = MD2("hello"); # returns "a9046c73e00331af68917d3804f70655"
```

Exceptions

MD2-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MD2_bin()

43.39.2.6 binary Qore::MD2_bin (data *data*)

Returns the MD2 message digest of the supplied argument as binary object.

Platform Availability:

Qore::Option::HAVE_MD2

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = MD2_bin("hello");
```

Exceptions

MD2-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MD2()

43.39.2.7 string Qore::MD4 (data *data*)

Returns the MD4 message digest of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "866437cb7a794bce2b727acc0362ee27")

Example:

```
my string $str = MD4("hello"); # returns "866437cb7a794bce2b727acc0362ee27"
```

Exceptions

MD4-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MD4_bin()

43.39.2.8 binary Qore::MD4_bin (data *data*)

Returns the MD4 message digest of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = MD4_bin("hello");
```

Exceptions

MD4-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MD4()

43.39.2.9 string Qore::MD5 (data *data*)

Returns the MD5 message digest of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "5d41402abc4b2a76b9719d911017c592")

Example:

```
my string $str = MD5("hello"); # returns "5d41402abc4b2a76b9719d911017c592"
```

Exceptions

MD5-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

- equivalent to Qore::zzz8stringzzz9::toMD5()
- the MD5 algorithm is not collision-resistant; it's recommended to use another hash algorithm (like SHA-256) if cryptographic security is important

See also

MD5_bin()

43.39.2.10 binary Qore::MD5_bin (data *data*)

Returns the MD5 message digest of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = MD5_bin("hello");
```

Exceptions

MD5-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MD5()

43.39.2.11 string Qore::MDC2 (data *data*)

Returns the MDC2 message digest of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_MDC2

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "4517036cf97b2407d6fe22aa5ab878a3")

Example:

```
my string $str = MDC2("hello"); # returns "4517036cf97b2407d6fe22aa5ab878a3"
```

Exceptions

MDC2-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MDC2_bin()

43.39.2.12 binary Qore::MDC2_bin (data *data*)

Returns the MDC2 message digest of the supplied argument as a binary object.

Platform Availability:

Qore::Option::HAVE_MDC2

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = MDC2_bin("hello");
```

Exceptions

MDC2-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

MDC2_bin()

43.39.2.13 string Qore::RIPEMD160 (data *data*)

Returns the RIPEMD message digest of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "108f07b8382412612c048d07d13f814118445acd")

Example:

```
my string $str = RIPEMD160("hello"); # returns "108f07b8382412612c048d07d13f814118445acd"
```

Exceptions

RIPEMD160-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

RIPEMD160_bin()

43.39.2.14 binary Qore::RIPEMD160_binary (data *data*)

Returns the RIPEMD message digest of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = RIPEMD160_bin("hello");
```

Exceptions

RIPEMD160-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

RIPEMD160()

43.39.2.15 string Qore::SHA (data *data*)

Returns the SHA (outdated SHA-0) message digest of the supplied argument as a hex string.

This hash algorithm was withdrawn after publishing and is considered to have serious flaws.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "ac62a630ca850b4ea07eda664eaecf9480843152")

Example:

```
my string $str = SHA("hello"); returns "ac62a630ca850b4ea07eda664eaecf9480843152"
```

Exceptions

SHA-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA_bin()

43.39.2.16 string Qore::SHA1 (data *data*)

Returns the SHA1 message digest of the supplied argument as a hex string.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d")

Example:

```
my string $str = SHA1("hello"); # "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

Exceptions

SHA1-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::zzz8stringzzz9::toSHA1()

See also

SHA1_bin()

43.39.2.17 binary Qore::SHA1_bin (data *data*)

Returns the SHA1 message digest of the supplied argument as a binary object.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA1_bin("hello");
```

Exceptions

SHA1-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA1()

43.39.2.18 string Qore::SHA224 (data *data*)

Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA224

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193")

Example:

```
my string $str = SHA224("hello"); # returns "ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193
      "
```

Exceptions

SHA224-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::zzz8stringzzz9::toSHA224()

See also

SHA224_bin()

43.39.2.19 binary Qore::SHA224_bin (data *data*)

Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a binary object.

Platform Availability:

Qore::Option::HAVE_SHA224

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA224_bin("hello");
```

Exceptions

SHA224-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA224()

43.39.2.20 string Qore::SHA256 (data *data*)

Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA256

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Example:

```
my string $str = SHA256("hello"); # returns "
      2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
```

Exceptions

SHA256-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::zzz8stringzzz9::toSHA256()

See also

SHA256_bin()

43.39.2.21 binary Qore::SHA256_bin (data *data*)

Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a binary object.

Platform Availability:

Qore::Option::HAVE_SHA256

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA256_bin("hello");
```

Exceptions

SHA256-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA256()

43.39.2.22 string Qore::SHA384 (data *data*)

Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA384

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90

Example:

```
my string $str = SHA384("hello"); # returns "
      59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90125a3c79f90397bdf5f6a13de828684f"
```

Exceptions

SHA384-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::zzz8stringzzz9::toSHA384()

See also

SHA384_bin()

43.39.2.23 binary Qore::SHA384_bin (data *data*)

Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a binary object.

Platform Availability:

Qore::Option::HAVE_SHA384

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA384_bin("hello");
```

Exceptions

SHA384-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA384()

43.39.2.24 string Qore::SHA512 (data *data*)

Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA512

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a hex string of the digest (ex: "9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca7

Example:

```
my string $str = SHA512("hello"); # returns "
      9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3d99ba5c11d7c7acc6e14b8c5da0c4663475c2e5c3adef
```

Exceptions

SHA512-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::zzz8stringzzz9::toSHA512()

See also

SHA512_bin()

43.39.2.25 binary Qore::SHA512_bin (data *data*)

Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a binary object.

Platform Availability:

Qore::Option::HAVE_SHA512

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA512_bin("hello");
```

Exceptions

SHA512-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA512()

43.39.2.26 binary Qore::SHA_bin (data *data*)

Returns the SHA (outdated SHA-0) message digest of the supplied argument as a binary object.

This hash algorithm was withdrawn after publishing and is considered to have serious flaws.

Code Flags:

RET_VALUE_ONLY

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

Returns

a binary object of the digest

Example:

```
my binary $bin = SHA_bin("hello");
```

Exceptions

SHA-DIGEST-ERROR	error calculating digest (should not normally happen)

See also

SHA()

43.40 HMAC Functions

Functions

- string Qore::DSS1_hmac (data data, string key)

  Returns the DSS1 (SHA-1 and DSA) based HMAC of the supplied argument as a hex string.
- string Qore::DSS_hmac (data data, string key)

  Returns the DSS (SHA-0 and DSA) based HMAC of the supplied argument as a hex string.
- string Qore::MD2_hmac (data data, string key)

  Returns the MD2 based HMAC of the supplied argument as a hex string.
- string Qore::MD4_hmac (data data, string key)

  Returns the MD4 based HMAC of the supplied argument as a hex string.
- string Qore::MD5_hmac (data data, string key)

  Returns the MD5 based HMAC of the supplied argument as a hex string.
- string Qore::MDC2_hmac (data data, string key)

  Returns the MDC2 based HMAC of the supplied argument as a hex string.
- string Qore::RIPEMD160_hmac (data data, string key)

  Returns the RIPEMD based HMAC of the supplied argument as a hex string.
- string Qore::SHA1_hmac (data data, string key)

  Returns the SHA1 based HMAC of the supplied argument as a hex string.
- string Qore::SHA224_hmac (data data, string key)

  Returns the SHA224 based HMAC of the supplied argument as a hex string.
- string Qore::SHA256_hmac (data data, string key)

  Returns the SHA256 based HMAC of the supplied argument as a hex string.
- string Qore::SHA384_hmac (data data, string key)

  Returns the SHA384 based HMAC of the supplied argument as a hex string.
- string Qore::SHA512_hmac (data data, string key)

  Returns the SHA512 based HMAC of the supplied argument as a hex string.
- string Qore::SHA_hmac (data data, string key)

  Returns the SHA based HMAC of the supplied argument as a hex string.

43.40.1 Detailed Description

Qore's cryptography support is provided by the OpenSSL library.

In cryptography, a keyed-hash message authentication code (HMAC) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authentication of a message. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

For more info: Wikipedia's Hash-based message authentication code article.

See also:

- Crytographic Functions
- Digest (Hash) Functions
- Crytographic Contants

43.40.2 Function Documentation

43.40.2.1 string Qore::DSS1_hmac (data *data,* string *key*)

Returns the DSS1 (SHA-1 and DSA) based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "37a3cc73159aa129b0eb22bbdf4b9309d389f629")

Example:

```
my string $str = DSS1_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "37a3cc73159aa129b0eb22bbdf4b9309d389f629"
```

Note

this algorithm is considered outdated and is included for backwards-compatibility

Exceptions

DSS1-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.2 string Qore::DSS_hmac (data *data*, string *key*)

Returns the DSS (SHA-0 and DSA) based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "37a3cc73159aa129b0eb22bbdf4b9309d389f629")

Example:

```
my string $str = DSS_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "37a3cc73159aa129b0eb22bbdf4b9309d389f629"
```

Note

this algorithm is considered outdated and is included for backwards-compatibility

Exceptions

DSS-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.3 string Qore::MD2_hmac (data *data*, string *key*)

Returns the MD2 based HMAC of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_MD2

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "27f5f17500b408e97643403ea8ef1413")

Example:

```
my string $str = MD2_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "27f5f17500b408e97643403ea8ef1413"
```

Exceptions

MD2-HMAC-ERROR	error calculating digest (should not normally happen)

See also

MD2_bin()

43.40.2.4 string Qore::MD4_hmac (data *data*, string *key*)

Returns the MD4 based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "053d084f321a3886e60166ebd9609ce1")

Example:

```
my string $str = MD4_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "053d084f321a3886e60166ebd9609ce1"
```

Exceptions

MD4-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.5 string Qore::MD5_hmac (data *data*, string *key*)

Returns the MD5 based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments

key	a secret passphrase/key

Returns

a hex string of the digest (ex: "87505c6164aaf6ca6315233902a01ef4")

Example:

```
my string $str = MD5_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "87505c6164aaf6ca6315233902a01ef4"
```

Note

the MD5 algorithm is not collision-resistant; it's recommended to use another hash algorithm (like SHA-256) if cryptographic security is important

Exceptions

MD5-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.6 string Qore::MDC2_hmac (data *data*, string *key*)

Returns the MDC2 based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "e0ef6a6803e58807c5db395e180a999c")

Example:

```
my string $str = MDC2_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "e0ef6a6803e58807c5db395e180a999c"
```

Exceptions

MDC2-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.7 string Qore::RIPEMD160_hmac (data *data*, string *key*)

Returns the RIPEMD based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "4bca70bca1601aba57624eeb2606535cb12f2079")

Example:

```
my string $str = RIPEMD160_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "4bca70bca1601aba57624eeb2606535cb12f2079"
```

Exceptions

RIPEMD160-HMAC-ERR↩ OR	error calculating digest (should not normally happen)

43.40.2.8 string Qore::SHA1_hmac (data *data,* string *key*)

Returns the SHA1 based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "37a3cc73159aa129b0eb22bbdf4b9309d389f629")

Example:

```
my string $str = SHA1_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "37a3cc73159aa129b0eb22bbdf4b9309d389f629"
```

Note

Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

Exceptions

SHA1-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.9 string Qore::SHA224_hmac (data *data,* string *key*)

Returns the SHA224 based HMAC of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA224

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "fad5667fa5aa412044555b7e077fced62372fe9c6ce20815609da12c")

Example:

```
my string $str = SHA224_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "fad5667fa5aa412044555b7e077fced62372fe9c6ce20815609da12c"
```

Exceptions

SHA224-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.10 string Qore::SHA256_hmac (data *data*, string *key*)

Returns the SHA256 based HMAC of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA256

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "1c90c21e227712b62019ff831f34cba22c2e70f1a902651ef69a70705ee0f754

Example:

```
my string $str = SHA256_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "1c90c21e227712b62019ff831f34cba22c2e70f1a902651ef69a70705ee0f754"
```

Exceptions

SHA256-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.11 string Qore::SHA384_hmac (data *data*, string *key*)

Returns the SHA384 based HMAC of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA384

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "e2c253c6dcb050990b4da3cee95cd7b227f43388fa8116f476f59395af295d0d

Example:

```
my string $str = SHA384_hmac("Hello There This is a Test - 1234567890", "a key");
# returns
       "e2c253c6dcb050990b4da3cee95cd7b227f43388fa8116f476f59395af295d0d3bb7156ab2fcd0663b0500249a7a0865"
```

Exceptions

SHA384-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.12 string Qore::SHA512_hmac (data *data*, string *key*)

Returns the SHA512 based HMAC of the supplied argument as a hex string.

Platform Availability:

Qore::Option::HAVE_SHA512

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "8dcefd7ea3f90ff1c822b5e9547fc36edf78c3e4ce13d47510a212a406bdda1a

Example:

```
my string $str = SHA512_hmac("Hello There This is a Test - 1234567890", "a key");
# returns
       "8dcefd7ea3f90ff1c822b5e9547fc36edf78c3e4ce13d47510a212a406bdda1a4094e7ea5ade90e1c736e204d331a814520eba49f3d074e2c
```

Exceptions

SHA512-HMAC-ERROR	error calculating digest (should not normally happen)

43.40.2.13 string Qore::SHA_hmac (data *data*, string *key*)

Returns the SHA based HMAC of the supplied argument as a hex string.

Parameters

data	the data to process and produce a digest for; the trailing null character is not included in the digest when processing string arguments
key	a secret passphrase/key

Returns

a hex string of the digest (ex: "0ad47c8d36dc4606d52f7e4cbd144ef2fda492a0")

Example:

```
my string $str = SHA_hmac("Hello There This is a Test - 1234567890", "a key");
# returns "0ad47c8d36dc4606d52f7e4cbd144ef2fda492a0"
```

Note

SHA/SHA0 was withdrawn shortly after publication due to an undisclosed "significant flaw" and replaced by the slightly revised version SHA-1.

Exceptions

SHA-HMAC-ERROR	error calculating digest (should not normally happen)

43.41 Crytographic Contants

Variables

- const Qore::DefaultIV = <0000000000000000>

  The default initialization vector is simply a 8-byte string of nulls.

43.41.1 Detailed Description

Cryptographic constants

43.42 Old DBI Functions

Functions

- __7_ int Qore::SQL::getDBIDriverCapabilities (string driver)

  Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or NOTHING if the driver is not already loaded.
- nothing Qore::SQL::getDBIDriverCapabilities ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ list Qore::SQL::getDBIDriverCapabilityList (string driver)

  Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.
- nothing Qore::SQL::getDBIDriverCapabilityList ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ list Qore::SQL::getDBIDriverList ()

  Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.
- hash Qore::SQL::parseDatasource (string ds)

  Returns a datasource hash of the components of a datasource string.
- nothing Qore::SQL::parseDatasource ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.42.1 Detailed Description

Old DBI functions; see DBI Functions

43.42.2 Function Documentation

43.42.2.1 __7_ int Qore::SQL::getDBIDriverCapabilities (string *driver*)

Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or NOTHING if the driver is not already loaded.

Code Flags:

CONSTANT

Parameters

driver	the name of the driver; if the given driver is not already loaded then this function returns NOTHING

Returns

an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or NOTHING if the driver is not already loaded

Example:

```
my *int $caps = getDBIDriverCapabilities("pgsql");
```

See also

dbi_get_driver_capabilities() for a similar function that uses the standard function naming scheme (ie "names_like_this()" instead of "camelCase()"), always returns an int, and does not have a N↩ OOP variant

43.42.2.2 nothing Qore::SQL::getDBIDriverCapabilities ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.42.2.3 __7_ list Qore::SQL::getDBIDriverCapabilityList (string *driver*)

Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.

Code Flags:

CONSTANT

Parameters

driver	the name of the driver; if the given driver is not loaded then the function returns NOTHING

Returns

a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found

Example:

```
my *list $l = getDBIDriverCapabilityList("pgsql");
```

See also

dbi_get_driver_capability_list() for a similar function that uses the standard function naming scheme (ie "`names_like_this()`" instead of "`camelCase()`") and does not have a NOOP variant

43.42.2.4 nothing Qore::SQL::getDBIDriverCapabilityList ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.42.2.5 __7_ list Qore::SQL::getDBIDriverList ( )

Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.

Returns

a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded

Code Flags:

CONSTANT

Example:

```
my *list $l = getDBIDriverList();
```

See also

dbi_get_driver_list() for a similar function that uses the standard function naming scheme (ie "names_↩like_this()" instead of "camelCase()")

43.42.2.6 hash Qore::SQL::parseDatasource (string *ds*)

Returns a datasource hash of the components of a datasource string.

Code Flags:

RET_VALUE_ONLY

Parameters

ds	a string describing the datasource with the following syntax: [*driver*:][*user*][:[/*pass*]]@*db*[(*charset*)][%*host*[:*port*][{*option*=*val*[,...]}]] where all elements except @*db* are optional

Returns

a datasource hash of the components of a datasource string

Example:

```
my hash $h = parseDatasource("
      pgsql:user/pass@dbname(utf8)%dbhost.internal:1521{min=4,max=10}");
```

Exceptions

DATASOURCE-PARSE-ERROR	a syntax error occurred parsing the datasource string (missing field, unexpected character, etc)

See also

parse_datasource() for a similar function that uses the standard function naming scheme (ie "names_↩like_this()" instead of "camelCase()") and does not have a NOOP variant

43.42.2.7 nothing Qore::SQL::parseDatasource ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.43 DBI Functions

Functions

- int Qore::SQL::dbi_get_driver_capabilities (string driver)

  Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or 0 if the driver is not already loaded.
- __7_ list Qore::SQL::dbi_get_driver_capability_list (string driver)

  Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.
- __7_ list Qore::SQL::dbi_get_driver_list ()

  Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.
- __7_ hash Qore::SQL::dbi_get_driver_options (string driver)

  returns a hash of driver options
- hash Qore::SQL::parse_datasource (string ds)

  Returns a datasource hash of the components of a datasource string.

43.43.1 Detailed Description

These DBI functions were added in Qore 0.8.6; the original functions used camel case names and had NOOP variants (see Old DBI Functions)

43.43.2 Datasource Hash

Key	Description
`type`	the name of the driver, if present
`user`	the username given in the string
`pass`	the password for the connection
`db`	the database name for the connection
`charset`	The name of the DB-specific character encoding to use for the connection, if present in the string
`host`	the hostname for the connection, if present in the string
`port`	the port number to use for the connection, if present in the string
`options`	A hash of options given in the string, if present. Special options are `"min"` and `"max"`, which are respected by the DatasourcePool::constructor(hash) variant for setting the minimum and maximum connections in the pool, respectively; other options are passed to the DBI driver

43.43.3 Function Documentation

43.43.3.1 int Qore::SQL::dbi_get_driver_capabilities (string *driver*)

Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or 0 if the driver is not already loaded.

Code Flags:

CONSTANT

Parameters

driver	the name of the driver; if the given driver is not already loaded then this function returns NOTHING

Returns

an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or 0 if the driver is not already loaded

Example:

```
my int $caps = dbi_get_driver_capabilities("pgsql");
```

Note

similar to getDBIDriverCapabilities() except uses the standard function naming scheme (ie "`names_like_this()`" instead of "`camelCase()`"), always returns an int, and does not have a NOOP variant

Since

Qore 0.8.6

43.43.3.2 __7_ list Qore::SQL::dbi_get_driver_capability_list (string *driver*)

Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.

Code Flags:

CONSTANT

Parameters

driver	the name of the driver; if the given driver is not loaded then the function returns NOTHING

Returns

a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found

Example:

```
my *list $l = dbi_get_driver_capability_list("pgsql");
```

Note

similar to getDBIDriverCapabilityList() except uses the standard function naming scheme (ie "`names_like_this()`" instead of "`camelCase()`") and does not have a NOOP variant

Since

Qore 0.8.6

43.43.3.3 __7_ list Qore::SQL::dbi_get_driver_list ( )

Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.

Returns

a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded

Code Flags:

CONSTANT

Example:

```
my *list $l = dbi_get_driver_list();
```

Note

similar to getDBIDriverList() except uses the standard function naming scheme (ie "`names_like_this()`" instead of "`camelCase()`")

Since

Qore 0.8.6

43.43.3.4 __7_ hash Qore::SQL::dbi_get_driver_options (string *driver*)

returns a hash of driver options

Code Flags:

CONSTANT

Parameters

driver	the name of the driver; if the given driver is not already loaded then this function returns NOTHING

Returns

if the given driver is not already loaded then the function returns NOTHING; if the driver does not support any options then an empty hash is returned, otherwise a hash is returned where the keys are valid option names, and the values are hashes with the following keys:

- "`desc`": a string description of the option
- "`type`": a string giving the data type restriction for the option

Example:

```
my *hash $h = dbi_get_driver_options("pgsql");
```

Since

Qore 0.8.6

43.43.3.5 hash Qore::SQL::parse_datasource (string *ds*)

Returns a datasource hash of the components of a datasource string.

Code Flags:

RET_VALUE_ONLY

Parameters

ds	a string describing the datasource with the following syntax: `[`*driver*`:][`*user*`:[/`*pass*`]@`*db*`[(`*charset*`)][%`*host*`[:`*port*`][{`*option*`=`*val*`[,...]}]` where all elements except `@`*db* are optional

Returns

a datasource hash of the components of a datasource string

Example:

```
my hash $h = parse_datasource("
      pgsql:user/pass@dbname(utf8)%dbhost.internal:1521{min=4,max=10}");
```

Exceptions

DATASOURCE-PARSE-ERROR	a syntax error occurred parsing the datasource string (missing field, unexpected character, etc)

Note

similar to parseDatasource() except uses the standard function naming scheme (ie "`names_like_this()`" instead of "`camelCase()`") and does not have a NOOP variant

Since

Qore 0.8.6

43.44 SQL Constants

Variables

- const Qore::SQL::BLOB = "blob"

  for binding BLOB values
- const Qore::SQL::CLOB = "clob"

  for binding CLOB values
- const Qore::SQL::DATE = "date"

  fod binding date/time values
- const Qore::SQL::DECIMAL = "number"

  for binding decimal values as a number
- const Qore::SQL::NUMBER = "number"

  for binding number values as a number
- const Qore::SQL::NUMERIC = "number"

  for binding numeric values as a number
- const Qore::SQL::VARCHAR = "string"

  for binding string values

43.44.1 Detailed Description

SQL constants

43.44.2 Variable Documentation

43.44.2.1 const Qore::SQL::DECIMAL = "number"

for binding decimal values as a number

Since

Qore 0.8.6 the value of this constant is `"number"` instead of `"string"`

43.44.2.2 const Qore::SQL::NUMBER = "number"

for binding number values as a number

Since

Qore 0.8.6 the value of this constant is `"number"` instead of `"string"`

43.44.2.3 const Qore::SQL::NUMERIC = "number"

for binding numeric values as a number

Since

Qore 0.8.6 the value of this constant is `"number"` instead of `"string"`

43.45 Environment Functions

Functions

- __7_ string Qore::getenv (string var)

  Retrieves the value of an environment variable or NOTHING if the variable is not set.
- nothing Qore::getenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::setenv (string env, softstring val)

  Sets an environment variable to a value.
- nothing Qore::setenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::unsetenv (string env)

  Unsets an environment variable.
- nothing Qore::unsetenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.45.1 Detailed Description

Environment functions

43.45.2 Function Documentation

43.45.2.1 __7_ string Qore::getenv (string *var*)

Retrieves the value of an environment variable or NOTHING if the variable is not set.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

var	the name of the environment variable

Returns

the value of an environment variable or NOTHING if the variable is not set

Example:

```
my *string $v = getenv("PATH");
```

Since

Qore 0.8.4 tagged with PO_NO_EXTERNAL_INFO

43.45.2.2 nothing Qore::getenv ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

NOOP

43.45.2.3 int Qore::setenv (string *env*, softstring *val*)

Sets an environment variable to a value.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

env	the name of the environment variable
val	the new value of the environment variable

Returns

-1 for error or 0 for success; if an error occurs, errno() and/or strerror() can be used to get the error code or message

Example:

```
if (setenv("PATH", "/bin:/usr/bin"))
    printf("error setting PATH: %s\n", strerror());
```

See also

unsetenv() to unset or clear an environment variable

Since

Qore 0.8.4 tagged with PO_NO_PROCESS

43.45.2.4 nothing Qore::setenv ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Code Flags:

NOOP

43.45.2.5 int Qore::unsetenv (string *env*)

Unsets an environment variable.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

env	the name of the environment variable to unset

Returns

-1 for error or 0 for success; if an error occurs, errno() and/or strerror() can be used to get the error code or message

Example:

```
if (unsetenv("PATH"))
    printf("error unsetting PATH: %s\n", strerror());
```

See also

setenv() to set an environment variable to a value

Since

Qore 0.8.4 tagged with PO_NO_PROCESS

43.45.2.6 nothing Qore::unsetenv ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

NOOP

43.46 Filesystem Functions

Functions

- int Qore::chdir (string path)

  Changes the current working directory for the current process.
- int Qore::chmod (string path, softint mode)

  Changes the mode of a file or directory.
- int Qore::chown (string path, softint owner=-1, softint group=-1)

  Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), follows symbolic links.
- string Qore::getcwd ()

  Returns a string giving the current working directory or NOTHING if the current working directory could not be read.
- string Qore::getcwd2 ()

  Returns a string giving the current working directory; throws an exception if the current directory cannot be read.
- __7_ list Qore::glob (string glob_str)

  Returns a list of files matching the string argument or NOTHING if the call to glob() fails.
- nothing Qore::glob ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::hlstat (string path)

  Returns a hash of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.
- nothing Qore::hlstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::hstat (string path)

  Returns a hash of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.
- nothing Qore::hstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::is_bdev (string path)

  Returns True if the string passed identifies a block device on the filesystem, False if not.
- bool Qore::is_cdev (string path)

  Returns True if the string passed identifies a character device on the filesystem, False if not.
- bool Qore::is_dev (string path)

  Returns True if the string passed identifies a device (either block or character) on the filesystem, False if not.
- bool Qore::is_dir (string path)

  Returns True if the string passed identifies a directory on the filesystem, False if not.
- bool Qore::is_executable (string path)

  Returns True if the string passed identifies an executable on the filesystem, False if not.
- bool Qore::is_file (string path)

  Returns True if the string passed identifies a regular file on the filesystem, False if not.
- bool Qore::is_link (string path)

  Returns True if the string passed identifies a symbolic link on the filesystem, False if not.
- bool Qore::is_pipe (string path)

  Returns True if the string passed identifies a pipe (FIFO) on the filesystem, False if not.
- bool Qore::is_readable (string path)

  Returns True if the string passed identifies a file readable by the current user, False if not.
- bool Qore::is_socket (string path)

  Returns True if the string passed identifies a socket on the filesystem, False if not.

- bool Qore::is_writable (string path)

  Returns True if the string passed identifies a file writable by the current user, False if not.

- bool Qore::is_writeable (string path)

  Returns True if the string passed identifies a file writable by the current user (backwards-compatible misspelling of is_writable())

- int Qore::lchown (string path, softint uid=-1, softint gid=-1)

  Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), does not follow symbolic links but rather operates on the symbolic link itself.

- __7_ list Qore::lstat (string path)

  Returns a list of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

- nothing Qore::lstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::mkdir (string path, softint mode=0777)

  Creates a directory, optionally specifying the mode.

- int Qore::mkfifo (string path, softint mode=0600)

  Creates a named pipe file with an optional file mode.

- string Qore::readlink (string path)

  Returns the target of a symbolic link; throws an exception if an error occurs (ex: file does not exist or is not a symbolic link)

- nothing Qore::rename (string old_path, string new_path)

  Renames (or moves) files or directories. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem objects or paths to execute the rename operation.

- int Qore::rmdir (string path)

  Removes a directory.

- __7_ list Qore::stat (string path)

  Returns a list of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

- nothing Qore::stat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash Qore::statvfs (string path)

  Returns a hash of filesystem status values for the file or directory path passed.

- nothing Qore::symlink (string old_path, string new_path)

  Creates a symbolic link to a directory path. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem path to create the symbolic link.

- int Qore::umask (softint mask)

  Sets the file creation mode mask for the process and returns the previous value of the file creation mode mask.

- nothing Qore::umask ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::unlink (string path)

  Deletes a file and returns 0 for success, -1 for error (in which case errno() can be used to get the error)

- nothing Qore::unlink ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.46.1 Detailed Description

The following functions return information about or are related to the filesystem.

All of the functions in this section (except umask()) are flagged with Qore::PO_NO_FILESYSTEM

See also

the File class for a class enabling files to be created, read or written, and the Dir class allowing directories to be manipulated

43.46.2 Filesystem Status Hash

Key	Description
`namemax`	The maximum length in mytes of file names on the filesystem
`fsid`	The filesystem ID; may not be set or meaningful on all filesystems/systems: see system documentation for statvfs()
`frsize`	The size in bytes of the minimum allocation unit on the filesystem
`bsize`	The filesystem's block size
`flag`	Flags describing mount options for the filesystem
`files`	The total number of inodes on the filesystem
`favail`	The number of free inodes available to unpriviledged users
`ffree`	The total number of free indes available to priviledged users
`blocks`	The total number of blocks on the filesystem (capacity in bytes = `bsize * blocks`)
`bavail`	The number of free blocks available to unpriviledged users (bytes = `bsize * bavail`)
`bfree`	The total number of free indes available to priviledged users (bytes = `bsize * bfree`)

43.46.3 Stat List

Element	Data Type	Description
0	int	device inode number the file is on
1	int	inode of the file
2	int	inode protection mode
3	int	number of hard links to this file
4	int	user ID of the owner
5	int	group ID of the owner
6	int	device type number
7	int	file size in bytes
8	date	last access time of the file
9	date	last modified time of the file
10	date	last change time of the file's inode
11	int	block size; may be zero if the platform's internal stat() (2) function does not provide this info

12	int	blocks allocated for the file; may be zero if the platform's internal stat() (2) function does not provide this info

43.46.4 Stat Hash

Key	Data Type	Description
dev	int	device inode number the file is on
inode	int	inode of the file
mode	int	inode protection mode
nlink	int	number of hard links to this file
uid	int	user ID of the owner
gid	int	group ID of the owner
rdev	int	device type number
size	int	file size in bytes
atime	date	last access time of the file
mtime	date	last modified time of the file
ctime	date	last change time of the file's inode
blksize	int	block size; may be zero if the platform's internal stat() (2) function does not provide this info
blocks	int	blocks allocated for the file; may be zero if the platform's internal stat() (2) function does not provide this info
type	string	a string giving the file type; one of: - "BLOCK-DEVICE" - "DIRECTORY" - "CHARACTER-DEVICE" - "FIFO" - "SYMBOLIC-LINK" - "SOCKET" - "REGULAR" - "UNKNOWN"
perm	string	a string giving UNIX-style permissions for the file (ex: "-rwxr-xr-x")

43.46.5 Function Documentation

43.46.5.1 int Qore::chdir (string *path*)

Changes the current working directory for the current process.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL, Qore::PO_NO_FILESYSTEM

Parameters

path	the new working directory for the current process

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (chdir($dir))
    printf("chdir %s: %s\n", $dir, strerror());
```

Since

Qore 0.8.4 this function is tagged with Qore::PO_NO_PROCESS_CONTROL

43.46.5.2 int Qore::chmod (string *path*, softint *mode*)

Changes the mode of a file or directory.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the file or directory to change
mode	the new mode for the file or directory

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (chmod("/bin/login", 0755))
    printf("rmdir /tmp/newdir: %s\n", strerror());
```

43.46.5.3 int Qore::chown (string *path*, softint *owner* = -1, softint *group* = -1)

Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), follows symbolic links.

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	path to the file or directory to change
owner	the uid of the new user owner; -1 means do not change user owner

group	the gid of the new group owner; -1 means do not change group owner

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (chown("/bin/login", 0, 0))
    printf("chown /bin/login: %s\n", strerror());
```

See also

lchown() for a version of this function that does not follow symbolic links

43.46.5.4 string Qore::getcwd ( )

Returns a string giving the current working directory or NOTHING if the current working directory could not be read.

Returns

a string giving the current working directory or NOTHING if the current working directory could not be read

Restrictions:

Qore::PO_NO_FILESYSTEM, Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Example:

```
my *string $cwd = getcwd();
```

See also

getcwd2() for a similar function that throws an exception if an error occurs instead

43.46.5.5 string Qore::getcwd2 ( )

Returns a string giving the current working directory; throws an exception if the current directory cannot be read.

Restrictions:

Qore::PO_NO_FILESYSTEM, Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $cwd = getcwd2();
```

See also

getcwd() for a similar function that returns NOTHING instead of throwing an exception if an error occurs

43.46.5.6 __7_ list Qore::glob (string *glob_str*)

Returns a list of files matching the string argument or NOTHING if the call to glob() fails.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

glob_str	the glob string, containing an optional path (in which case the entire path must be readable) and a glob filename pattern

Returns

a list of files matching the string argument or NOTHING if the call to glob() fails, in which case errno() can be used to get the error

Example:

```
my *list $gl = glob("*.txt");
if (!exists $gl)
    printf("failed to glob *.txt: %s\n", strerror());
```

43.46.5.7 nothing Qore::glob ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.46.5.8 __7_ hash Qore::hlstat (string *path*)

Returns a hash of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to retrieve information for

Returns

a hash of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number

Example:

```
my *hash $h = hlstat("/tmp/file.txt");
if (!exists $h)
    printf("could not hlstat /tmp/file.txt: %s\n", strerror());
```

See also

hstat() for a version of this function that follows symbolic links

lstat() for a version of this function that returns a traditional list instead of a user-friendly hash

File::hlstat() for a static method in the File class that throws an exception instead of returning NOTHING when errors occur

43.46.5.9 nothing Qore::hlstat ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.46.5.10 __7_ hash Qore::hstat (string *path*)

Returns a hash of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to retrieve information for

Returns

a hash of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number

Example:

```
my *hash $h = hstat("/tmp/file.txt");
if (!exists $h)
    printf("could not hstat /tmp/file.txt: %s\n", strerror());
```

See also

hlstat() for a version of this function that does not follow symbolic links

stat() for a version of this function that returns a traditional list instead of a user-friendly hash

File::hstat() for a static method in the File class that throws an exception instead of returning NOTHING when errors occur

43.46.5.11 nothing Qore::hstat ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.46.5.12 bool Qore::is_bdev (string *path*)

Returns True if the string passed identifies a block device on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a block device on the filesystem, False if not

Example:

```
my bool $b = is_bdev("/tmp/sda1");
```

43.46.5.13 bool Qore::is_cdev (string *path*)

Returns True if the string passed identifies a character device on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a character device on the filesystem, False if not

Example:

```
my bool $b = is_cdev("/tmp/tty");
```

43.46.5.14 bool Qore::is_dev (string *path*)

Returns True if the string passed identifies a device (either block or character) on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a device (either block or character) on the filesystem, False if not

Example:

```
my bool $b = is_dev("/tmp/scanner");
```

43.46.5.15 bool Qore::is_dir (string *path*)

Returns True if the string passed identifies a directory on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a directory on the filesystem, False if not

Example:

```
my bool $b = is_dir("/tmp/mydir");
```

43.46.5.16 bool Qore::is_executable (string *path*)

Returns True if the string passed identifies an executable on the filesystem, False if not.

Platform Availability

Qore::Option::HAVE_IS_EXECUTABLE

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies an executable on the filesystem, False if not

Example:

```
my bool $b = is_executable("/bin/login");
```

43.46.5.17 bool Qore::is_file (string *path*)

Returns True if the string passed identifies a regular file on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a regular file on the filesystem, False if not

Example:

```
my bool $b = is_file("/etc/hosts");
```

43.46.5.18 bool Qore::is_link (string *path*)

Returns True if the string passed identifies a symbolic link on the filesystem, False if not.

Platform Availability

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a symbolic link on the filesystem, False if not

Example:

```
my bool $b = is_link("/etc/hosts");
```

43.46.5.19 bool Qore::is_pipe (string *path*)

Returns True if the string passed identifies a pipe (FIFO) on the filesystem, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a pipe (FIFO) on the filesystem, False if not

Example:

```
my bool $b = is_pipe("/tmp/mypipe");
```

43.46.5.20 bool Qore::is_readable (string *path*)

Returns True if the string passed identifies a file readable by the current user, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a file readable by the current user, False if not

Example:

```
my bool $b = is_readable("/etc/hosts");
```

Bug on Windows this function always returns True for directories; this will be fixed in a later version of Qore

43.46.5.21 bool Qore::is_socket (string *path*)

Returns True if the string passed identifies a socket on the filesystem, False if not.

Platform Availability

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a socket on the filesystem, False if not

Example:

```
my bool $b = is_socket("/tmp/X0");
```

43.46.5.22 bool Qore::is_writable (string *path*)

Returns True if the string passed identifies a file writable by the current user, False if not.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a file writable by the current user, False if not

Example:

```
my bool $b = is_writable("/etc/hosts");
```

Bug on Windows this function always returns False for directories; this will be fixed in a later version of Qore

43.46.5.23 bool Qore::is_writeable (string *path*)

Returns True if the string passed identifies a file writable by the current user (backwards-compatible misspelling of is_writable())

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

CONSTANT

Parameters

path	the path to check

Returns

True if the string passed identifies a file writable by the current user, False if not

Example:

```
my bool $b = is_writeable("/etc/hosts");
```

43.46.5.24 int Qore::lchown (string *path*, softint *uid* = -1, softint *gid* = -1)

Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), does not follow symbolic links but rather operates on the symbolic link itself.

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	path to the file or directory to change
uid	the uid of the new user owner; -1 means do not change user owner
gid	the gid of the new group owner; -1 means do not change group owner

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (lchown("/tmp/socket", 0, 0))
    printf("lchown /tmp/socket: %s\n", strerror());
```

See also

chown() for a version of this function that follows symbolic links (ie operates on the target instead of on the link itself)

43.46.5.25 __7_ list Qore::lstat (string *path*)

Returns a list of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to retrieve information for

Returns

a list of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number

Example:

```
my *list $l = lstat("/tmp/file.txt");
if (!exists $l)
    printf("could not lstat /tmp/file.txt: %s\n", strerror());
```

Note

on platforms without symbolic links (such as with native Windows ports, for example), this function is identical to stat()

See also

stat() for a version of this function that follows symbolic links

hlstat() for a version of this function that returns a user-friendly hash instead of a list

File::lstat() for a static method in the File class that throws an exception instead of returning NOTHING when errors occur

43.46.5.26 nothing Qore::lstat ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.46.5.27 int Qore::mkdir (string *path*, softint *mode* = 0777)

Creates a directory, optionally specifying the mode.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the directory to create
mode	the file mode of the new directory (which will be AND'ed with the umask)

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (mkdir("/tmp/newdir", 0755))
    printf("mkdir /tmp/newdir: %s\n", strerror());
```

See also

umask()

43.46.5.28 int Qore::mkfifo (string *path*, softint *mode* = 0600)

Creates a named pipe file with an optional file mode.

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the new named pipe
mode	the file mode for the new named pipe

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (mkfifo("/tmp/pipe"))
    printf("mkfifo /tmp/pipe: %s\n", strerror());
```

43.46.5.29 string Qore::readlink (string *path*)

Returns the target of a symbolic link; throws an exception if an error occurs (ex: file does not exist or is not a symbolic link)

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the symbolic link

Returns

the target of the link; throws an exception if the given path is not a symbolic link (or if another error occurs)

Example:

```
my string $str = readlink("/tmp/symbolic_link");
```

Exceptions

READLINK-ERROR	Invalid arguments or a system error occured (ex: file does not exist or is not a symbolic link)

43.46.5.30 nothing Qore::rename (string *old_path*, string *new_path*)

Renames (or moves) files or directories. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem objects or paths to execute the rename operation.

This function does not return any value; if any errors occur, an exception is thrown.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

old_path	The original path for the file to move
new_path	The target path for the file

Example:

```
rename("/tmp/myfile", "/tmp/myfile.txt");
```

Exceptions

RENAME-ERROR	empty string passed for one of the arguments or the operating system returned an error

Note

Some operating systems do not allow moving files between filesystems (ex Solaris)

43.46.5.31 int Qore::rmdir (string *path*)

Removes a directory.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the directory to remove

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (rmdir("/tmp/newdir"))
    printf("rmdir /tmp/newdir: %s\n", strerror());
```

43.46.5.32 __7_ list Qore::stat (string *path*)

Returns a list of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to retrieve information for

Returns

a list of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number

Example:

```
my *list $l = stat("/tmp/file.txt");
if (!exists $l)
    printf("could not stat /tmp/file.txt: %s\n", strerror());
```

See also

lstat() for a version of this function that does not follow symbolic links

hstat() for a version of this function that returns a user-friendly hash instead of a list

File::stat() for a static method in the File class that throws an exception instead of returning NOTHING when errors occur

43.46.5.33 nothing Qore::stat ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.46.5.34 __7_ hash Qore::statvfs (string *path*)

Returns a hash of filesystem status values for the file or directory path passed.

Platform Availability:

Qore::Option::HAVE_STATVFS

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the filesystem (or on the filesystem)

Returns

a Filesystem Status Hash; or, if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number

Example:

```
my *hash $h = statvfs("/tmp")
```

See also

File::statvfs() for a static method in the File class that throws an exception instead of returning NOTHING when errors occur

43.46.5.35 nothing Qore::symlink (string *old_path,* string *new_path*)

Creates a symbolic link to a directory path. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem path to create the symbolic link.

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Restrictions:

Qore::PO_NO_FILESYSTEM

This function does not return any value; if any errors occur, an exception is thrown. If the target of the symbolic link does not exist, it is not an error; the symbolic link is created anyway.

Parameters

old_path	The original path; the target of the link
new_path	The path to the location of the new symbolic link to be created with this function call

Example:

```
symlink("/tmp/temporary-dir", "/users/oracle/install");
```

Exceptions

SYMLINK-ERROR	empty string passed for one of the arguments or the operating system returned an error

Since

Qore 0.8.5

43.46.5.36 int Qore::umask (softint *mask*)

Sets the file creation mode mask for the process and returns the previous value of the file creation mode mask.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

mask	the new file creation mode mask for the process

Returns

the old file creation mode mask for the process

Example:

```
if (unlink($path))
    printf("%s: %s\n", $path, strerror());
```

43.46.5.37 nothing Qore::umask ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Code Flags:

RUNTIME_NOOP

43.46.5.38 int Qore::unlink (string *path*)

Deletes a file and returns 0 for success, -1 for error (in which case errno() can be used to get the error)

Does not delete directories; see rmdir() for a similar function that removes directories

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

path	the path to the file to delete

Returns

0 for success, -1 for error (in which case errno() can be used to get the error)

Example:

```
if (unlink($path))
    printf("%s: %s\n", $path, strerror());
```

43.46.5.39 nothing Qore::unlink ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_FILESYSTEM

Code Flags:

RUNTIME_NOOP

43.47 Library Functions

Functions

- nothing Qore::abort ()

  Aborts the current program (this function does not return)
- string Qore::basename (string path)

  Returns a string giving the last element of a file path (meant to be the filename)
- nothing Qore::basename ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing Qore::close_all_fd (__7_ softbool strd)

  closes all possible file descriptors; useful in "daemon" processes that may have inherited open file descriptors
- string Qore::dirname (string path)

  Returns a string giving the path up to a file but not the filename itself.
- nothing Qore::dirname ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::errno ()

  Returns the error code of the last error that occurred in the current thread.
- nothing Qore::exec (string command)

  Replaces the current process image with another; this function does not return.
- nothing Qore::exit (softint rc=0)

  Exits the program with the return code passed (this function does not return)
- int Qore::fork ()

  Creates a copy of the current process with a new PID; returns 0 in the child process; returns the child's PID in the parent process.
- list Qore::getaddrinfo (__7_ string node, __7_ softstring service, softint family=AF_UNSPEC, softint flags=0)

  Returns a list of Address Information Hash for the given node name or string address; if no lookup can be performed then an exception is thrown.
- int Qore::getegid ()

  Returns the effective group ID of the current process.
- int Qore::geteuid ()

  Returns the effective user ID of the current process.
- int Qore::getgid ()

  Returns the real group ID of the current process.
- list Qore::getgroups ()

  returns a list of group IDs that the user is a member of
- __7_ string Qore::gethostbyaddr (string addr, softint type=AF_INET)

  Returns the official hostname corresponding to the network addressed passed as an argument.
- nothing Qore::gethostbyaddr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::gethostbyaddr_long (string addr, softint type=AF_INET)

  Returns a hash representing all host and address information corresponding to the address and address type passed as arguments.
- nothing Qore::gethostbyaddr_long ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ string Qore::gethostbyname (string name)

  Returns the first address corresponding to the hostname passed as an argument or NOTHING if the lookup fails.
- nothing Qore::gethostbyname ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash Qore::gethostbyname_long (string name)

  Returns a hash representing all host and address information corresponding to the hostname passed as an argument.
- nothing Qore::gethostbyname_long ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::gethostname ()

  Returns the hostname of the system.
- int Qore::getpid ()

  Returns the PID (process ID) of the current process.
- int Qore::getppid ()

  Returns the PID (process ID) of the parent process of the current process.
- int Qore::getuid ()

  Returns the real user ID of the current process.
- int Qore::kill (softint pid, softint sig=SIGHUP)

  Sends a signal to a process, if no signal number is given, then Qore::SIGHUP is sent by default.
- nothing Qore::kill ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::rand ()

  Returns a random 64-bit integer number.
- int Qore::setegid (softint gid)

  Changes the process effective group ID according to the argument passed.
- int Qore::seteuid (softint uid)

  Changes the effective process user ID according to the argument passed.
- int Qore::setgid (softint gid)

  Changes the process group ID according to the argument passed.
- Qore::setgroups (softlist gids)

  sets the list of supplementary group IDs for the current process
- int Qore::setsid ()

  Creates a new session lead by the calling process.
- int Qore::setuid (softint uid)

  Changes the process user ID according to the argument passed.
- int Qore::sleep (softint seconds)

  Causes the current thread to sleep for a certain number of seconds.
- nothing Qore::sleep ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing Qore::srand (softint seed)

  Seeds the random number generator with the integer passed.
- nothing Qore::srand ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::strerror (softint err)

  Returns the string corresponding to the error code passed (generally retrieved with errno())
- string Qore::strerror ()

  Returns the string corresponding to the last error that occurred in the current thread.
- int Qore::system (string command)

  executes a command and returns the exit code of the process
- nothing Qore::system ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::usleep (softint usecs)

  Causes the current thread to sleep for a certain number of microseconds.

- int Qore::usleep (date d)

  Causes the current thread to sleep for a certain number of microseconds.

- nothing Qore::usleep ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.47.1 Detailed Description

Library functions

43.47.2 Host Information Hash

Key	Type	Description
`name`	string	The official fully-qualified name of the host
`aliases`	list of string	Any hostname aliases for the host
`typename`	string	The type of network address (either `"ipv4"` or `"ipv6"`)
`type`	int	One of the Network Address Family Constants (either Qore::AF_INET or Qore::AF_INET6) corresponding to the type of network addresses given
`len`	int	The length of the addresses in bytes when represented in binary form
`addresses`	list of string	All addresses corresponding to the host; the list should have at least 1 element

43.47.3 Address Information Hash

Key	Type	Description
`address`	string	A valid address of the host, for example: `"::1"`.
`address_desc`	string	A descriptive string of the address containing the address family, for example: `"ipv6[::1]"`
`family`	int	The network address family; see Network Address Family Constants.
`familystr`	string	A descriptive string for the network address family, for example: `"ipv6"`.

addrlen	int	The length of the internal network address data structure (not normally needed in Qore but provided anyway)
[port]	int	The port number corresponding to the service (if applicable)

43.47.4 Function Documentation

43.47.4.1 nothing Qore::abort ( )

Aborts the current program (this function does not return)

This function causes the current process to terminate abnormally; a core dump or crash report may be generated if enabled

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Example:

```
abort();
```

See also

exit()

43.47.4.2 string Qore::basename (string *path*)

Returns a string giving the last element of a file path (meant to be the filename)

Code Flags:

CONSTANT

Parameters

path	the path to process

Returns

a string giving the last element of the given file path (meant to be the filename)

Example:

```
my string $fn = basename("/usr/local/bin/file_name");
```

See also

dirname()

43.47.4.3 nothing Qore::basename ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.4 nothing Qore::close_all_fd (__7_ softbool *strd*)

closes all possible file descriptors; useful in "daemon" processes that may have inherited open file descriptors

Platform Availability:

Qore::Option::HAVE_CLOSE_ALL_FD

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

strd	if True then also stdin, stdout, and stderr are closed, otherwise fds $>$ 2 are closed

Example:

```
close_all_fd();
```

Note

if there are file descriptors open by Qore when this function is called, this function will also close those file descriptors; normally this function should only be used when starting a new process after a fork() and exec() to close any inherited descriptors, for example

43.47.4.5 string Qore::dirname (string *path*)

Returns a string giving the path up to a file but not the filename itself.

Code Flags:

CONSTANT

Parameters

path	the path to process

Returns

a string giving the path up to a file but not the filename itself; if no directory separator characters can be found in the path, **"."** is returned (meaning the current directory)

Example:

```
my string $dir = dirname("/usr/local/bin/file_name");
```

See also

basename()

43.47.4.6 nothing Qore::dirname ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.7 int Qore::errno ( )

Returns the error code of the last error that occurred in the current thread.

Returns

the error code of the last error that occurred in the current thread (see Error Constants for possible error code values)

Example:

```
if (unlink($path))
    printf("%s: %s\n", $path, strerror(errno()));
```

See also

strerror() for a function that gives the string description for the error number returned by this function

43.47.4.8 nothing Qore::exec (string *command*)

Replaces the current process image with another; this function does not return.

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS, Qore::PO_NO_PROCESS_CONTROL

Parameters

command	the executable to execute and any command-line arguments; the command is executed internally by a call to execvp() (3)

Example:

```
exec("/usr/bin/xterm -bg black -fg white -sb -sl 2000");
```

43.47.4.9 nothing Qore::exit (softint *rc* = 0)

Exits the program with the return code passed (this function does not return)

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

rc	the return code for the process (0 = no error; success)

Example:

```
exit(0);
```

See also

abort()

43.47.4.10 int Qore::fork ( )

Creates a copy of the current process with a new PID; returns 0 in the child process; returns the child's PID in the parent process.

This function will throw an `ILLEGAL-FORK` exception if more than one thread is running

Platform Availability:

Qore::Option::HAVE_FORK

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Returns

0 in the child process; returns the child's PID in the parent process; if an error occurs, then -1 is returned, in this case no child process was started and the error number can be retrieved with the errno() function

Example:

```
my int $pid;
if ($pid = fork())
    printf("child has PID %d\n", $pid);
```

Exceptions

ILLEGAL-FORK	Cannot fork if more than one thread is running

Note

- after a fork(), the only safe call is exec(), many system calls are not async-signal safe, including pthread_create() which is used to start new threads.
- the Qore process will crash if unsafe operations are called after a fork()
- the signal-handling thread cannot be reliably started on many platforms (ex: FreeBSD) after a fork(), therefore signal handling is disabled in the child process after a fork()
- on Darwin (OS/X) threading primitives are unusable in the child process after a fork()

43.47.4.11 list Qore::getaddrinfo (__7_ string *node*, __7_ softstring *service*, softint *family* = `AF_UNSPEC`, softint *flags* = `0`)

Returns a list of Address Information Hash for the given node name or string address; if no lookup can be performed then an exception is thrown.

Either node or service may be NOTHING but not both or a `QOREADDRINFO-GETINFO-ERROR` exception will be thrown

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Parameters

node	The host name or node name to look up
service	The service name to look up to a port number
family	The address family for the lookup, must be one of Qore::AF_INET, Qore::AF_INET6, or Qore::AF_UNSPEC, meaning to return all possible addresses
flags	see Network Address Information Constants for possible values to be combined with binary or

Returns

a list of Address Information Hash for the given node name or string address; if no lookup can be performed then an exception is thrown

Example:

```
my list $l = getaddrinfo("localhost");
```

Exceptions

QOREADDRINFO-GETINFO-ERROR	nodename nor servname provided, or not known

43.47.4.12 int Qore::getegid ( )

Returns the effective group ID of the current process.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Returns

the effective group ID of the current process

Example:

```
my int $egid = getegid();
```

43.47.4.13 int Qore::geteuid ( )

Returns the effective user ID of the current process.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Returns

the effective user ID of the current process

Example:

```
my int $euid = geteuid();
```

43.47.4.14 int Qore::getgid ( )

Returns the real group ID of the current process.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Returns

the real group ID of the current process

Example:

```
my int $gid = getgid();
```

43.47.4.15 list Qore::getgroups ( )

returns a list of group IDs that the user is a member of

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Example:

```
my list $l = getgroups();
```

Returns

a list of group IDs that the user is a member of

Since

Qore 0.8.11.1

43.47.4.16 __7_ string Qore::gethostbyaddr (string *addr*, softint *type* = AF_INET)

Returns the official hostname corresponding to the network addressed passed as an argument.

If the address family is invalid or the address string is not a valid address for the given family a GETHOSTBYADD↩R-ERROR exception will be thrown.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

addr	the address to look up
type	the network address family; see Network Address Family Constants for valid values

Returns

the official hostname corresponding to the network addressed passed as an argument

Example:

```
my *string $hostname = gethostbyaddr("192.168.0.33");
if (!exists $hostname)
    printf("address lookup on 192.168.0.33 failed\n", $host);
```

Exceptions

GETHOSTBYADDR-ERR↩ OR	invalid address for the given family or invalid address family

See also

gethostbyaddr_long() for a version of this function that returns all host information, including all hostname aliases and all addresses

43.47.4.17 nothing Qore::gethostbyaddr ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.18 __7_ hash Qore::gethostbyaddr_long (string *addr,* softint *type* = AF_INET)

Returns a hash representing all host and address information corresponding to the address and address type passed as arguments.

If the address family is invalid or the address string is not a valid address for the given family a GETHOSTBYADD↩ R-ERROR exception will be thrown.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

addr	the address to look up
type	the network address family; see Network Address Family Constants for valid values

Returns

a hash representing all host and address information corresponding to the address and address type passed as arguments

Example:

```
my *hash $ah = gethostbyaddr_long("192.168.0.33");
if (!exists $ah)
    printf("address lookup on 192.168.0.33 failed\n", $host);
```

Exceptions

GETHOSTBYADDR-ERR↩ OR	invalid address for the given family or invalid address family

See also

gethostbyaddr() for a simpler version of this function that returns just a single hostname for the address

43.47.4.19 nothing Qore::gethostbyaddr_long ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.20 __7_ string Qore::gethostbyname (string *name*)

Returns the first address corresponding to the hostname passed as an argument or NOTHING if the lookup fails.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

name	the name to look up

Returns

the first address corresponding to the hostname passed as an argument or NOTHING if the lookup fails

Example:

```
my *string $addr = gethostbyname($host);
if (!exists $host)
    printf("address lookup on %y failed; hostname unknown\n", $host);
```

See also

gethostbyname_long() for a version of this function that returns all host information, including all hostname aliases and all addresses

43.47.4.21 nothing Qore::gethostbyname ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.22 __7_ hash Qore::gethostbyname_long (string *name*)

Returns a hash representing all host and address information corresponding to the hostname passed as an argument.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

name	the name to look up

Returns

a hash representing all host and address information corresponding to the hostname passed as an argument

Example:

```
my *hash $ah = gethostbyname_long($host);
if (!exists $host)
    printf("address lookup on %y failed; hostname unknown\n", $host);
```

See also

gethostbyname() for a version of this function that returns just the first network address corresponding to the hostname

43.47.4.23 nothing Qore::gethostbyname_long ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.47.4.24 string Qore::gethostname ( )

Returns the hostname of the system.

Returns

the hostname of the system

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Example:

```
my string $host = gethostname();
```

43.47.4.25 int Qore::getpid ( )

Returns the PID (process ID) of the current process.

Returns

the PID (process ID) of the current process

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $pid = getpid();
```

43.47.4.26 int Qore::getppid ( )

Returns the PID (process ID) of the parent process of the current process.

Platform Availability:

Qore::Option::HAVE_GETPPID

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Returns

the PID (process ID) of the parent process of the current process

Example:

```
my int $ppid = HAVE_GETPPID ? getppid() : -1;
```

43.47.4.27 int Qore::getuid ( )

Returns the real user ID of the current process.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Returns

the real user ID of the current process

Example:

```
my int $uid = getuid();
```

43.47.4.28 int Qore::kill (softint *pid*, softint *sig* = SIGHUP)

Sends a signal to a process, if no signal number is given, then Qore::SIGHUP is sent by default.

Platform Availability:

Qore::Option::HAVE_KILL

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Parameters

pid	the pid argument can have the following interpretations: • > 0: the signal is sent to the process ID given • 0: the signal is sent to all processes whose group ID is equal to the process group ID of the sender, and for which the process has permission • −1: if the user has super-user privileges, the signal is sent to all processes excluding system processes and the process sending the signal. If the user is not the super user, the signal is sent to all processes with the same uid as the user, excluding the process sending the signal. No error is returned if any process could be signaled.

sig	the signal number to send to the process or processes

Returns

0 means success (no error), or -1 meaning an error occurred; in this case check errno() for the error

Example:

```
if (kill($pid, SIGTERM))
    printf("error sending signal to pid %d: %s\n", $pid, strerror());
```

43.47.4.29 nothing Qore::kill ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Code Flags:

RUNTIME_NOOP

43.47.4.30 int Qore::rand ( )

Returns a random 64-bit integer number.

This function uses the C library function random() to generate the number if available on the current platform, otherwise uses rand().

Returns

a random 64-bit integer number

Code Flags:

CONSTANT

Example:

```
my int $num = rand();
```

See also

srand() for a function to seed the random number generator

43.47.4.31 int Qore::setegid (softint *gid*)

Changes the process effective group ID according to the argument passed.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

gid	the group ID to set

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (setegid($gid))
    printf("setegid %d: %s\n", $gid, strerror());
```

43.47.4.32 int Qore::seteuid (softint *uid*)

Changes the effective process user ID according to the argument passed.

Platform Availability:

Qore::Option::HAVE_SETEUID

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

uid	the user ID to set

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (seteuid($uid))
    printf("seteuid %d: %s\n", $uid, strerror());
```

43.47.4.33 int Qore::setgid (softint *gid*)

Changes the process group ID according to the argument passed.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

gid	the group ID to set

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (setgid($gid))
    printf("setgid %d: %s\n", $gid, strerror());
```

43.47.4.34 Qore::setgroups (softlist *gids*)

sets the list of supplementary group IDs for the current process

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Example:

```
setgroups($l);
```

Parameters

gids	a list of supplementary group IDs for the current process

Since

Qore 0.8.11.1

43.47.4.35 int Qore::setsid ( )

Creates a new session lead by the calling process.

The calling process is the session leader of the new session, is the process group leader of a new process group and has no controlling terminal. The calling process is the only process in either the session or the process group.

Returns

Upon successful completion, the setsid() function returns the value of the process group ID of the new process group, which is the same as the process ID of the calling process; if an error occurs, setsid() returns -1 and errno() and/or strerror() can be used to check the error.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Example:

```
if (setsid() == -1)
    printf("setsid(): %s\n", strerror());
```

43.47.4.36 int Qore::setuid (softint *uid*)

Changes the process user ID according to the argument passed.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

uid	the user ID to set

Returns

0 if no error occurred; -1 if an error occurred, in which case errno() and/or strerror() can be used to retrieve the error

Example:

```
if (setuid($uid))
    printf("setuid %d: %s\n", $uid, strerror());
```

43.47.4.37 int Qore::sleep (softint *seconds*)

Causes the current thread to sleep for a certain number of seconds.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

seconds	The amount of time in seconds to sleep; integer arguments are interpreted literally as a number of seconds to sleep, however a relative date/time value can be given instead of an integer to make the source more readable (ex: `5s`), however as this function only supports a resolution of 1 second, milliseconds and microseconds are ignored if passed in a relative date/time value

Returns

should always return 0 for success

Example:

```
sleep(10s);
```

See also

usleep() for a similar function supporting microsecond resolution

43.47.4.38 nothing Qore::sleep ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Code Flags:

RUNTIME_NOOP

43.47.4.39 nothing Qore::srand (softint *seed*)

Seeds the random number generator with the integer passed.

This function uses the C library function srandom() if available on the current platform, otherwise uses srand()

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

seed	the seed for the random number generator (only the least-significant 32-bits are used)

Example:

```
srand(now());
```

43.47.4.40 nothing Qore::srand ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Code Flags:

RUNTIME_NOOP

43.47.4.41 string Qore::strerror (softint *err*)

Returns the string corresponding to the error code passed (generally retrieved with errno())

Code Flags:

CONSTANT

Parameters

err	the error code to retrieve the description for (generally provided by errno())

Returns

the string corresponding to the error code passed; if the error code is unknown, then a string like "Unknown error: -1" is returned

Example:

```
if (unlink($path))
    printf("%s: %s\n", $path, strerror(errno()));
```

43.47.4.42 string Qore::strerror ( )

Returns the string corresponding to the last error that occurred in the current thread.

Returns

the string corresponding to the last error that occurred in the current thread

Code Flags:

CONSTANT

Example:

```
if (unlink($path))
    printf("%s: %s\n", $path, strerror());
```

Since

Qore 0.8.4 this variant of the function automatically uses the last error value for the current thread

43.47.4.43 int Qore::system (string *command*)

executes a command and returns the exit code of the process

Platform Availability:

Qore::Option::HAVE_SYSTEM

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Parameters

command	the command to execute; if shell meta-characters are found (currently defined as any of $=*?\><";) or if the current platform does not support fork(), then the command is executed with system() (3), otherwise fork() and execvp() (3) are used instead

Returns

the exit code of the process executed

Example:

```
int rc = system("ls -l");
```

See also

- backquote()
- the backquote operator

43.47.4.44 nothing Qore::system ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Code Flags:

RUNTIME_NOOP

43.47.4.45 int Qore::usleep (softint *usecs*)

Causes the current thread to sleep for a certain number of microseconds.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

usecs	The amount of time in microseconds to sleep; integer arguments are interpreted literally as a number of microseconds to sleep

Returns

should always return 0 for success

Example:

```
usleep(5000);
```

See also

sleep() for a similar function supporting second resolution

Note

both sleep() and usleep() are implemented internally by calls to nanosleep() (2) if available on the current platform, otherwise both are implemented by calls to usleep() (3)

43.47.4.46 int Qore::usleep (date *d*)

Causes the current thread to sleep for a certain number of microseconds.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

d	A time value giving the amount of time in microseconds to sleep; this should be a relative date/time value to make the source more readable (ex: `250ms`)

Returns

should always return 0 for success

Example:

```
usleep(250ms);
```

See also

sleep() for a similar function supporting second resolution

Note

both sleep() and usleep() are implemented internally by calls to nanosleep() (2) if available on the current platform, otherwise both are implemented by calls to usleep() (3)

43.47.4.47 nothing Qore::usleep ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Code Flags:

RUNTIME_NOOP

43.48 List Functions

Functions

- bool Qore::inlist ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::inlist (any arg, nothing x)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::inlist (any arg, softlist l)

  Returns True if the first argument is a member of the second argument list using soft comparisons (with implicit type conversions), False if not.
- bool Qore::inlist_hard ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::inlist_hard (any arg, nothing x)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::inlist_hard (any arg, softlist l)

  Returns True if the first argument is a member of the second argument list using hard comparisons (without any implicit type conversions), False if not.
- any Qore::max (list l)

  Returns the maximum value in a list.
- any Qore::max (list l, string func)

  Returns the maximum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.
- any Qore::max (list l, code f)

  Returns the maximum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.
- any Qore::max (...)

  Returns the maximum value of the arguments passed to the function.
- any Qore::min (list l)

  Returns the minumum value in a list.
- any Qore::min (list l, string func)

  Returns the minumum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.
- any Qore::min (list l, code f)

  Returns the minumum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.
- any Qore::min (...)

  Returns the minumum value of the arguments passed to the function.
- list Qore::range (int start, int stop, int step=1)

  Returns a list containing an arithmetic progression of integers.
- list Qore::range (int stop)

  Returns a list containing an arithmetic progression of integers with start = 0 and step = 1.
- nothing Qore::reverse ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- list Qore::reverse (list l)

  Reverses a list and returns the new list.
- any Qore::sort (any arg)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::sort (list l)

  Performs an unstable sort in ascending order and returns the new list.

- list Qore::sort (list l, string func)

  Performs an unstable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

- list Qore::sort (list l, code f)

  Performs an unstable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

- any Qore::sortDescending (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::sortDescending (list l)

  Performs an unstable sort in descending order and returns the new list.

- list Qore::sortDescending (list l, string func)

  Performs an unstable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

- list Qore::sortDescending (list l, code f)

  Performs an unstable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

- any Qore::sortDescendingStable (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::sortDescendingStable (list l)

  Performs a stable sort in descending order and returns the new list.

- list Qore::sortDescendingStable (list l, string func)

  Performs a stable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

- list Qore::sortDescendingStable (list l, code f)

  Performs a stable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

- any Qore::sortStable (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::sortStable (list l)

  Performs a stable sort in ascending order and returns the new list.

- list Qore::sortStable (list l, string func)

  Performs a stable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

- list Qore::sortStable (list l, code f)

  Performs a stable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

43.48.1 Detailed Description

List functions

43.48.2 Function Documentation

43.48.2.1 bool Qore::inlist ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.2 bool Qore::inlist (any *arg*, nothing *x*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.3 bool Qore::inlist (any *arg*, softlist *l*)

Returns True if the first argument is a member of the second argument list using soft comparisons (with implicit type conversions), False if not.

Code Flags:

RET_VALUE_ONLY

Example:

```
if (inlist($str, $strlist))
    printf("%y is in %y\n", $str, $strlist);
```

Parameters

arg	the argument to look for in the list
l	the list to search for the first argument *arg*

Returns

True if the first argument is a member of the second argument list using soft comparisons (with implicit type conversions), False if not

See also

inlist_hard(any, softlist)

43.48.2.4 bool Qore::inlist_hard ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.5 bool Qore::inlist_hard (any *arg*, nothing *x*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.6 bool Qore::inlist_hard (any *arg*, softlist *l*)

Returns True if the first argument is a member of the second argument list using hard comparisons (without any implicit type conversions), False if not.

Code Flags:

RET_VALUE_ONLY

Example:

```
if (inlist_hard($str, $strlist))
    printf("%y is in %y\n", $str, $strlist);
```

Parameters

arg	the argument to look for in the list
l	the list to search for the first argument *arg*

Returns

True if the first argument is a member of the second argument list using hard comparisons (without any implicit type conversions), False if not

See also

inlist(any, softlist)

43.48.2.7 any Qore::max (list *l*)

Returns the maximum value in a list.

This variant will only work on basic data types

Code Flags:

CONSTANT

Example:

```
my any $v = max($l);
```

Parameters

l	the list to process

Returns

the maximum value in a list

See also

min(list)

43.48.2.8 any Qore::max (list *l*, string *func*)

Returns the maximum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my any $v = max($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the maximum value in a list

See also

min(list, string)

43.48.2.9 any Qore::max (list *l*, code *f*)

Returns the maximum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my any $v = max($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the call reference or a closure must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the maximum value in a list

See also

min(list, code)

43.48.2.10 any Qore::max (...)

Returns the maximum value of the arguments passed to the function.

This variant will only work on basic data types

Code Flags:

CONSTANT

Example:

```
my any $v = max($v1, $v2, $v3);
```

Parameters

...	the list of values to process given directly to the function

Returns

the maximum value in a list

See also

min(...)

43.48.2.11 any Qore::min (list *l*)

Returns the minumum value in a list.

This variant will only work on basic data types

Code Flags:

CONSTANT

Example:

```
my any $v = min($l);
```

Parameters

l	the list to process

Returns

the minumum value in a list

See also

max(list)

43.48.2.12 any Qore::min (list *l*, string *func*)

Returns the minumum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my any $v = min($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the minumum value in a list

See also

max(list, string)

43.48.2.13 any Qore::min (list *l*, code *f*)

Returns the minumum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my any $v = min($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the call reference or a closure must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the minumum value in a list

See also

max(list, code)

43.48.2.14 any Qore::min (...)

Returns the minumum value of the arguments passed to the function.

This variant will only work on basic data types

Code Flags:

CONSTANT

Example:

```
my any $v = min($v1, $v2, $v3);
```

Parameters

...	the list of values to process given directly to the function

Returns

the minumum value in a list

See also

max(...)

43.48.2.15 list Qore::range (int *start*, int *stop*, int *step* = 1)

Returns a list containing an arithmetic progression of integers.

Code Flags:

RET_VALUE_ONLY

Example:

```
range(2, 5); # (2, 3, 4, 5)
range(2, -2); # (2, 1, 0, -1, -2)
range(1, 10, 5); # (1, 6)
range(0, 10, 5); # (0, 5, 10)
range(-10, 10, 5); # (-10, -5, 0, 5, 10)
range(10, -10, 5); # (10, 5, 0, -5, -10)
```

Parameters

start	the initial value
stop	the final value
step	the step; the default is 1; must be greater than 0; the function throws a `RANGE-ERROR` exception when this argument is < 1

Returns

Returns a list containing an arithmetic progression of integers.

Exceptions

RANGE-ERROR	this exception is thrown if `step` < 1

See also

xrange

Note

the main difference between range() and xrange() is that range returns real list and xrange returns a RangeIterator

Since

Qore 0.8.6

43.48.2.16 list Qore::range (int *stop*)

Returns a list containing an arithmetic progression of integers with start = 0 and step = 1.

This is an overloaded version of range(int, int, int) meaning `range`(0, `stop`, 1)

Code Flags:

CONSTANT

Example:

```
range(1); # (0, 1)
range(-3); # (0, -1, -2, -3)
```

Parameters

stop	the final value

Returns

Returns a list containing an arithmetic progression of integers with start = 0 and step = 1.

See also

xrange

Note

the main difference between range() and xrange() is that range returns real list and xrange returns a RangeIterator

Since

Qore 0.8.6

43.48.2.17 nothing Qore::reverse ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.18 list Qore::reverse (list *l*)

Reverses a list and returns the new list.

Code Flags:

CONSTANT

Example:

```
my list $nl = reverse($l);
```

Parameters

l	the list to reverse

Returns

the given list with all elements in reverse order

See also

reverse(string)

43.48.2.19 any Qore::sort (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.20 list Qore::sort (list *l*)

Performs an unstable sort in ascending order and returns the new list.

This variant will only sort basic data types

Code Flags:

CONSTANT

Example:

```
my list $nl = sort($l);
```

Parameters

l	the list to sort

Returns

the sorted list

See also

- sortStable(list)
- sortDescendingStable(list)
- sortDescending(list)

43.48.2.21 list Qore::sort (list *l*, string *func*)

Performs an unstable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my list $nl = sort($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, string)
- sortDescendingStable(list, string)
- sortDescending(list, string)

43.48.2.22 list Qore::sort (list *l*, code *f*)

Performs an unstable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my list $nl = sort($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the code must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, code)
- sortDescendingStable(list, code)
- sortDescending(list, code)

43.48.2.23 any Qore::sortDescending (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.24 list Qore::sortDescending (list *l*)

Performs an unstable sort in descending order and returns the new list.

This variant will only sort basic data types

Code Flags:

CONSTANT

Example:

```
my list $nl = sortDescending($l);
```

Parameters

l	the list to sort

Returns

the sorted list

See also

- sort(list)
- sortDescendingStable(list)
- sortStable(list)

43.48.2.25 list Qore::sortDescending (list *l*, string *func*)

Performs an unstable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

Code Flags:

CONSTANT

Example:

```
my list $nl = sortDescending($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, string)
- sortDescendingStable(list, string)
- sort(list, string)

43.48.2.26 list Qore::sortDescending (list *l*, code *f*)

Performs an unstable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my list $nl = sortDescending($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the code must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, code)
- sortDescendingStable(list, code)
- sort(list, code)

43.48.2.27 any Qore::sortDescendingStable (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.28 list Qore::sortDescendingStable (list *l*)

Performs a stable sort in descending order and returns the new list.

This variant will only sort basic data types

Code Flags:

CONSTANT

Example:

```
my list $nl = sortDescendingStable($l);
```

Parameters

l	the list to sort

Returns

the sorted list

See also

- sort(list)
- sortDescendingStable(list)
- sortStable(list)

43.48.2.29 list Qore::sortDescendingStable (list *l*, string *func*)

Performs a stable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my list $nl = sortDescending($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, string)
- sortDescendingStable(list, string)
- sort(list, string)

43.48.2.30 list Qore::sortDescendingStable (list *l*, code *f*)

Performs a stable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my list $nl = sortDescending($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the code must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sortStable(list, code)
- sortDescendingStable(list, code)
- sort(list, code)

43.48.2.31 any Qore::sortStable (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.48.2.32 list Qore::sortStable (list *l*)

Performs a stable sort in ascending order and returns the new list.

This variant will only sort basic data types

Code Flags:

CONSTANT

Example:

```
my list $nl = sortStable($l);
```

Parameters

l	the list to sort

Returns

the sorted list

See also

- sort(list)
- sortDescendingStable(list)
- sortDescending(list)

43.48.2.33 list Qore::sortStable (list *l*, string *func*)

Performs a stable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my list $nl = sortStable($l, "my_sort_function");
```

Parameters

l	the list to sort
func	the name of a function accessible in the current scope that accepts 2 arguments of the data type in the list; the function must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sort(list, string)
- sortDescendingStable(list, string)
- sortDescending(list, string)

43.48.2.34 list Qore::sortStable (list *l*, code *f*)

Performs a stable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.

Code Flags:

RET_VALUE_ONLY

Example:

```
my code $sort_func = int sub (hash $l, hash $r) { return $l.id <=> $r.id; };
my list $nl = sortStable($l, $sort_func);
```

Parameters

l	the list to sort
f	a call reference or a closure that accepts 2 arguments of the data type in the list; the code must return -1, 0, or 1 if the first is less than the second, if the first and second are equal, or if the first is greater than the second, respectively

Returns

the sorted list

See also

- sort(list, code)
- sortDescendingStable(list, code)
- sortDescending(list, code)

43.49 Math Functions

Functions

- int Qore::abs (int i)

  Returns the absolute value of the argument passed.
- number Qore::abs (number n)

  Returns the absolute value of the argument passed.
- float Qore::abs (softfloat f)

  Returns the absolute value of the argument passed.
- float Qore::abs ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::acos (number n)

  Returns the value in radians of the arc cosine of the given value.
- float Qore::acos (softfloat f)

  Returns the value in radians of the arc cosine of the given value.
- number Qore::asin (number n)

  Returns the value in radians of the arc sine of the given value.
- float Qore::asin (softfloat f)

  Returns the value in radians of the arc sine of the given value.
- float Qore::asin ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::atan (number n)

  Returns the value in radians of the arc tangent of the given value.
- float Qore::atan (softfloat f)

  Returns the value in radians of the arc tangent of the given value.
- float Qore::atan ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::atan2 (number y, number x)

  Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.
- float Qore::atan2 (softfloat y, softfloat x)

  Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.
- float Qore::atan2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::cbrt (number n)

  Returns the cube root of the number passed.
- float Qore::cbrt (softfloat f)

  Returns the cube root of the number passed.
- float Qore::cbrt ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::ceil (number n)

  Returns a number equal to the smallest integral value greater than or equal to the argument passed.
- float Qore::ceil (softfloat f)

  Returns a floating-point number equal to the smallest integral value greater than or equal to the argument passed.
- float Qore::ceil ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::cos (number n)

  Returns the cosine of the number in radians passed.
- float Qore::cos (float f)

  Returns the cosine of the number in radians passed.
- float Qore::cos ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::cosh (number n)

  Returns the hyperbolic cosine of the given value.
- float Qore::cosh (softfloat f)

  Returns the hyperbolic cosine of the given value.
- float Qore::cosh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::exp (number n)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number.
- float Qore::exp (softfloat f)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number.
- float Qore::exp ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::exp2 (number n)

  Returns the value of 2 raised to the power of the given number.
- float Qore::exp2 (softfloat f)

  Returns the value of 2 raised to the power of the given number.
- float Qore::exp2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::expm1 (number n)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number - 1.
- float Qore::expm1 (softfloat f)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number - 1.
- float Qore::expm1 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::floor (softnumber n)

  Returns a number equal to the largest integral value less than or equal to the argument passed.
- float Qore::floor (softfloat f)

  Returns a floating-point number equal to the largest integral value less than or equal to the argument passed.
- float Qore::floor ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::hypot (number x, number y)

  Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.
- float Qore::hypot (softfloat x, softfloat y)

  Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.
- float Qore::hypot ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number Qore::log10 (number n)

Returns the base 10 logarithm of the given number.

- float Qore::log10 (softfloat f)

  Returns the base 10 logarithm of the given number.

- float Qore::log10 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::log1p (number n)

  Returns the natural logarithm of 1 + the given number.

- float Qore::log1p (softfloat f)

  Returns the natural logarithm of 1 + the given number.

- float Qore::log1p ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- float Qore::logb (softfloat f)

  Returns the exponent of the given number.

- float Qore::logb ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::nlog (number n)

  Returns the natural logarithm of the given value.

- float Qore::nlog (softfloat f)

  Returns the natural logarithm of the given value.

- float Qore::nlog ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::pow (number x, number y)

  Returns a number raised to the power of another number.

- float Qore::pow (softfloat x=0.0, softfloat y=0.0)

  Returns a number raised to the power of another number.

- number Qore::round (number n)

  Returns a number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.

- float Qore::round (softfloat f)

  Returns a floating-point number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.

- float Qore::round ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::sin (number n)

  Returns the sine of the number in radians passed.

- float Qore::sin (softfloat f)

  Returns the sine of the number in radians passed.

- float Qore::sin ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::sinh (number n)

  Returns the hyperbolic sine of the given value.

- float Qore::sinh (softfloat f)

  Returns the hyperbolic sine of the given value.

- float Qore::sinh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::sqrt (number n)

Returns the square root of the number passed.

- float Qore::sqrt (softfloat f)

  Returns the square root of the number passed.

- float Qore::sqrt ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::tan (number n)

  Returns the tangent of the number in radians passed.

- float Qore::tan (softfloat f)

  Returns the tangent of the number in radians passed.

- float Qore::tan ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number Qore::tanh (number n)

  Returns the hyperbolic tangent of the given value.

- float Qore::tanh (softfloat f)

  Returns the hyperbolic tangent of the given value.

- float Qore::tanh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.49.1 Detailed Description

Math functions

43.49.2 Function Documentation

43.49.2.1 int Qore::abs (int *i*)

Returns the absolute value of the argument passed.

Code Flags:

CONSTANT

Parameters

i	the value to process

Returns

the absolute value of the argument passed

Example:

```
my int $i = abs(-20); # returns 20
```

Note

equivalent to Qore::zzz8intzzz9::abs()

43.49.2.2 number Qore::abs (number *n*)

Returns the absolute value of the argument passed.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the absolute value of the argument passed

Example:

```
my number $n = abs(-20n); # returns 20n
```

Note

equivalent to Qore::zzz8numberzzz9::abs()

43.49.2.3 float Qore::abs (softfloat *f*)

Returns the absolute value of the argument passed.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the absolute value of the argument passed

Example:

```
my float $f = abs(-20.2); # returns 20.2
```

Note

equivalent to Qore::zzz8floatzzz9::abs()

43.49.2.4 float Qore::abs ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.5 number Qore::acos (number *n*)

Returns the value in radians of the arc cosine of the given value.

Code Flags:

CONSTANT

Parameters

n	the cosine value to process

Returns

the value in radians of the arc cosine of the given value

Example:

```
my number $x = acos($y);
```

43.49.2.6 float Qore::acos (softfloat *f*)

Returns the value in radians of the arc cosine of the given value.

Code Flags:

CONSTANT

Parameters

f	the cosine value to process

Returns

the value in radians of the arc cosine of the given value

Example:

```
my float $x = acos($y);
```

43.49.2.7 number Qore::asin (number *n*)

Returns the value in radians of the arc sine of the given value.

Code Flags:

CONSTANT

Parameters

n	the sine value to process

Returns

the value in radians of the arc sine of the given value

Example:

```
my number $x = asin($y);
```

43.49.2.8 float Qore::asin (softfloat *f*)

Returns the value in radians of the arc sine of the given value.

Code Flags:

CONSTANT

Parameters

f	the sine value to process

Returns

the value in radians of the arc sine of the given value

Example:

```
my float $x = asin($y);
```

43.49.2.9 float Qore::asin ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.10 number Qore::atan (number *n*)

Returns the value in radians of the arc tangent of the given value.

Code Flags:

CONSTANT

Parameters

n	the tangent value to process

Returns

the value in radians of the arc tangent of the given value

Example:

```
my number $x = atan($y);
```

43.49.2.11 float Qore::atan (softfloat *f*)

Returns the value in radians of the arc tangent of the given value.

Code Flags:

CONSTANT

Parameters

f	the tangent value to process

Returns

the value in radians of the arc tangent of the given value

Example:

```
my float $x = atan($y);
```

43.49.2.12 float Qore::atan ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.13 number Qore::atan2 (number *y*, number *x*)

Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.

Code Flags:

CONSTANT

Parameters

y	the y value for the function
x	the x value for the function

Returns

the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result

Example:

```
my number $f = atan2($y, $x);
```

43.49.2.14 float Qore::atan2 (softfloat *y*, softfloat *x*)

Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.

Code Flags:

CONSTANT

Parameters

y	the y value for the function
x	the x value for the function

Returns

the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result

Example:

```
my float $f = atan2($y, $x);
```

43.49.2.15 float Qore::atan2 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.16 number Qore::cbrt (number *n*)

Returns the cube root of the number passed.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the cube root of the number passed

Example:

```
my number $x = cbrt($y);
```

43.49.2.17 float Qore::cbrt (softfloat *f*)

Returns the cube root of the number passed.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the cube root of the number passed

Example:

```
my float $x = cbrt($y);
```

43.49.2.18 float Qore::cbrt ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.19 number Qore::ceil (number *n*)

Returns a number equal to the smallest integral value greater than or equal to the argument passed.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

a number equal to the smallest integral value greater than or equal to the argument passed

Example:

```
my number $x = ceil(3.2n); # returns 4.0n
```

See also

floor(number)

43.49.2.20 float Qore::ceil (softfloat *f*)

Returns a floating-point number equal to the smallest integral value greater than or equal to the argument passed.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

a floating-point number equal to the smallest integral value greater than or equal to the argument passed

Example:

```
my float $x = ceil(3.2); # returns 4.0
```

See also

floor(softfloat)

43.49.2.21 float Qore::ceil ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.22 number Qore::cos (number *n*)

Returns the cosine of the number in radians passed.

Code Flags:

CONSTANT

Parameters

n	the angle in radians

Returns

the cosine of the number in radians passed

Example:

```
my number $x = cos($y);
```

43.49.2.23 float Qore::cos (float *f*)

Returns the cosine of the number in radians passed.

Code Flags:

CONSTANT

Parameters

f	the angle in radians

Returns

the cosine of the number in radians passed

Example:

```
my float $x = cos($y);
```

43.49.2.24 float Qore::cos ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.25 number Qore::cosh (number *n*)

Returns the hyperbolic cosine of the given value.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the hyperbolic cosine of the given value

Example:

```
my number $x = cosh($y);
```

43.49.2.26 float Qore::cosh (softfloat *f*)

Returns the hyperbolic cosine of the given value.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the hyperbolic cosine of the given value

Example:

```
my float $x = cosh($y);
```

43.49.2.27 float Qore::cosh ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.28 number Qore::exp (number *n*)

Returns the value of *e* (the base of natural logarithms) raised to the power of the given number.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the value of *e* (the base of natural logarithms) raised to the power of the given number

Example:

```
my number $x = exp($y);
```

43.49.2.29 float Qore::exp (softfloat *f*)

Returns the value of *e* (the base of natural logarithms) raised to the power of the given number.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the value of *e* (the base of natural logarithms) raised to the power of the given number

Example:

```
my float $x = exp($y);
```

43.49.2.30 float Qore::exp ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.31 number Qore::exp2 (number *n*)

Returns the value of 2 raised to the power of the given number.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the value of 2 raised to the power of the given number

Example:

```
my number $x = exp2($y);
```

43.49.2.32 float Qore::exp2 (softfloat *f*)

Returns the value of 2 raised to the power of the given number.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the value of 2 raised to the power of the given number

Example:

```
my float $x = exp2($y);
```

43.49.2.33 float Qore::exp2 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.34 number Qore::expm1 (number *n*)

Returns the value of *e* (the base of natural logarithms) raised to the power of the given number - 1.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the value of *e* (the base of natural logarithms) raised to the power of the given number - 1

Example:

```
my number $x = expm1($y);
```

43.49.2.35 float Qore::expm1 (softfloat *f*)

Returns the value of *e* (the base of natural logarithms) raised to the power of the given number - 1.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the value of *e* (the base of natural logarithms) raised to the power of the given number - 1

Example:

```
my float $x = expm1($y);
```

43.49.2.36 float Qore::expm1 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.37 number Qore::floor (softnumber *n*)

Returns a number equal to the largest integral value less than or equal to the argument passed.

Code Flags:

CONSTANT

Parameters

n	the value to process

Example:

```
my number $x = floor(3.9n); # returns 3.0n
```

See also

ceil(number)

43.49.2.38 float Qore::floor (softfloat *f*)

Returns a floating-point number equal to the largest integral value less than or equal to the argument passed.

Code Flags:

CONSTANT

Parameters

f	the value to process

Example:

```
my float $x = floor(3.9); # returns 3.0
```

See also

ceil(softfloat)

43.49.2.39 float Qore::floor ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.40 number Qore::hypot (number *x*, number *y*)

Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.

Code Flags:

CONSTANT

Parameters

x	the length of side x of the right-angle triangle
y	the length of side y of the right-angle triangle

Returns

the length of the hypotenuse of a right-angle triangle with sides given as the two arguments

Example:

```
my number $z = hypot($x, $y);
```

43.49.2.41 float Qore::hypot (softfloat *x*, softfloat *y*)

Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.

Code Flags:

CONSTANT

Parameters

x	the length of side x of the right-angle triangle
y	the length of side y of the right-angle triangle

Returns

the length of the hypotenuse of a right-angle triangle with sides given as the two arguments

Example:

```
my float $z = hypot($x, $y);
```

43.49.2.42 float Qore::hypot ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.43 number Qore::log10 (number *n*)

Returns the base 10 logarithm of the given number.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the base 10 logarithm of the given value

Example:

```
my number $x = log10($y);
```

43.49.2.44 float Qore::log10 (softfloat *f*)

Returns the base 10 logarithm of the given number.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the base 10 logarithm of the given value

Example:

```
my float $x = log10($y);
```

43.49.2.45 float Qore::log10 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.46 number Qore::log1p (number *n*)

Returns the natural logarithm of 1 + the given number.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the natural logarithm of 1 + the given number

Example:

```
my number $x = log1p($y);
```

43.49.2.47 float Qore::log1p (softfloat *f*)

Returns the natural logarithm of 1 + the given number.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the natural logarithm of 1 + the given number

Example:

```
my float $x = log1p($y);
```

43.49.2.48 float Qore::log1p ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.49 float Qore::logb (softfloat *f*)

Returns the exponent of the given number.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the exponent of the given number

Example:

```
my float $x = logb($y);
```

43.49.2.50 float Qore::logb ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.51 number Qore::nlog (number *n*)

Returns the natural logarithm of the given value.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the natural logarithm of the given value

Example:

```
my float $x = nlog($y);
```

43.49.2.52 float Qore::nlog (softfloat *f*)

Returns the natural logarithm of the given value.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the natural logarithm of the given value

Example:

```
my float $x = nlog($y);
```

43.49.2.53 float Qore::nlog ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.54 number Qore::pow (number *x*, number *y*)

Returns a number raised to the power of another number.

Code Flags:

RET_VALUE_ONLY

Parameters

x	the number to raise to the power of *y*
y	the power to raise *x* to

Returns

a number raised to the power of another number

Example:

```
my number $z = pow($x, $y);
```

Exceptions

DIVISION-BY-ZERO	in pow(x, y), if x = 0 then y must be a non-negative value

43.49.2.55 float Qore::pow (softfloat *x* = 0.0, softfloat *y* = 0.0)

Returns a number raised to the power of another number.

Code Flags:

RET_VALUE_ONLY

Parameters

x	the number to raise to the power of *y*
y	the power to raise *x* to

Returns

a number raised to the power of another number

Example:

```
my float $z = pow($x, $y);
```

Exceptions

DIVISION-BY-ZERO	in pow(x, y), if x = 0 then y must be a non-negative value
INVALID-POW-ARGUMENTS	in pow(x, y), x cannot be negative when y is not an integer value

43.49.2.56 number Qore::round (number *n*)

Returns a number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.

Code Flags:

CONSTANT

Parameters

n	a number to round

Returns

a number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero

Example:

```
my number $n = round($num);
```

43.49.2.57 float Qore::round (softfloat *f*)

Returns a floating-point number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.

Platform Availability

Qore::Option::HAVE_ROUND

Code Flags:

CONSTANT

Parameters

f	a number to round

Returns

a floating-point number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero

Example:

```
my float $n = round($num);
```

43.49.2.58 float Qore::round ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.59 number Qore::sin (number *n*)

Returns the sine of the number in radians passed.

Code Flags:

CONSTANT

Parameters

n	the angle in radians

Returns

the sine of the number in radians passed

Example:

```
my number $x = sin($y);
```

43.49.2.60 float Qore::sin (softfloat *f*)

Returns the sine of the number in radians passed.

Code Flags:

CONSTANT

Parameters

f	the angle in radians

Returns

the sine of the number in radians passed

Example:

```
my float $x = sin($y);
```

43.49.2.61 float Qore::sin ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.62 number Qore::sinh (number *n*)

Returns the hyperbolic sine of the given value.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the hyperbolic sine of the given value

Example:

```
my number $x = sinh($y);
```

43.49.2.63 float Qore::sinh (softfloat *f*)

Returns the hyperbolic sine of the given value.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the hyperbolic sine of the given value

Example:

```
my float $x = sinh($y);
```

43.49.2.64 float Qore::sinh ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.65 number Qore::sqrt (number *n*)

Returns the square root of the number passed.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the square root of the number passed

Example:

```
my number $x = sqrt($y);
```

43.49.2.66 float Qore::sqrt (softfloat *f*)

Returns the square root of the number passed.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the square root of the number passed

Example:

```
my float $x = sqrt($y);
```

43.49.2.67 float Qore::sqrt ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.68 number Qore::tan (number *n*)

Returns the tangent of the number in radians passed.

Code Flags:

CONSTANT

Parameters

n	the angle in radians

Returns

the tangent of the number in radians passed

Example:

```
my number $x = tan($y);
```

43.49.2.69 float Qore::tan (softfloat *f*)

Returns the tangent of the number in radians passed.

Code Flags:

CONSTANT

Parameters

f	the angle in radians

Returns

the tangent of the number in radians passed

Example:

```
my float $x = tan($y);
```

43.49.2.70 float Qore::tan ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.49.2.71 number Qore::tanh (number *n*)

Returns the hyperbolic tangent of the given value.

Code Flags:

CONSTANT

Parameters

n	the value to process

Returns

the hyperbolic tangent of the given value

Example:

```
my number $x = tanh($y);
```

43.49.2.72 float Qore::tanh (softfloat *f*)

Returns the hyperbolic tangent of the given value.

Code Flags:

CONSTANT

Parameters

f	the value to process

Returns

the hyperbolic tangent of the given value

Example:

```
my float $x = tanh($y);
```

43.49.2.73 float Qore::tanh ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.50 Math Constants

Variables

- const Qore::MAXINT = LLONG_MAX

  largest integer
- const Qore::MININT = qore(new QoreBigIntNode(-LLONG_MAX - 1))

  smallest integer
- const Qore::M_PI = 3.14159265358979323846

  PI (floating-point)
- const Qore::M_PIn = qore(pi_number())

  PI (arbitrary-precision numeric)

43.50.1 Detailed Description

Math constants

43.50.2 Variable Documentation

43.50.2.1 const Qore::M_PIn = qore(pi_number())

PI (arbitrary-precision numeric)

Since

Qore 0.8.6

43.50.2.2 const Qore::MAXINT = LLONG_MAX

largest integer

Since

Qore 0.8.6

43.50.2.3 const Qore::MININT = qore(new QoreBigIntNode(-LLONG_MAX - 1))

smallest integer

Since

Qore 0.8.6

43.51 Signal Handing Functions

Functions

- nothing Qore::remove_signal_handler (softint signal)

  Removes a signal handler and returns the signal handling state to the default.
- nothing Qore::set_signal_handler (softint signal, code f)

  Sets or replaces a signal handler according to the signal number and closure or call reference (function or object method reference) passed.

43.51.1 Detailed Description

Signal handing functions

43.51.2 Function Documentation

43.51.2.1 nothing Qore::remove_signal_handler (softint *signal*)

Removes a signal handler and returns the signal handling state to the default.

Platform Availability:

Qore::Option::HAVE_SIGNAL_HANDLING

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

By the time this function returns, changes to the signal handling thread have already been effected.

Parameters

signal	The signal number to process, see Signal Constants for possible values

Example:

```
remove_signal_handler(SIGINT);
```

See also

Signal Handling for more information

43.51.2.2 nothing Qore::set_signal_handler (softint *signal*, code *f*)

Sets or replaces a signal handler according to the signal number and closure or call reference (function or object method reference) passed.

Platform Availability:

Qore::Option::HAVE_SIGNAL_HANDLING

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

By the time this function returns, changes to the signal handling thread have already been effected.

When a signal is raised and the signal handler code is called, the signal number is passed as an integer argument to the signal handling code.

Parameters

signal	The signal number to process, see Signal Constants for possible values
f	The code to execute when the signal is caught; this should accept an integer argument giving the signal number

Example:

```
set_signal_handler(SIGINT, \signal_handler());
```

See also

Signal Handling for more information

43.52 Miscellaneous Functions

Functions

- string Qore::backquote (string cmd, __7_ reference rc)

  Executes a process and returns a string of the output (stdout only)
- nothing Qore::backquote ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- any Qore::call_builtin_function (string name,...)

  Calls a function and returns the return value, passing the remaining arguments after the function name to the builtin function.
- any Qore::call_builtin_function_args (string name, __7_ softlist vargs)

  Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.
- any Qore::call_function (string name,...)

  Calls a function and returns the return value, passing the remaining arguments after the function name to the function.
- any Qore::call_function (code f,...)

  Calls the given call reference or closure and returns the result, passing the remaining arguments to the call reference or closure.
- any Qore::call_function_args (string name, __7_ softlist vargs)

  Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.
- any Qore::call_function_args (code f, __7_ softlist vargs)

  Calls the given call reference or closure and returns the result, using the optional second argument as a list of arguments to the call reference or closure.
- string Qore::decode_url (string url)

  Decodes percent numeric codes in a URL string and returns the decoded string in UTF-8 encoding.
- nothing Qore::decode_url ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::encode_url (string url, softbool encode_all=False)

  Encodes URLs by substituting `'%'` characters with `'%25'`, spaces (`' '`) with `'%20'`, and non-ascii characters by percent-encoded representations.
- bool Qore::exists (...)

  A function performing the same role as the exists operator.
- bool Qore::existsFunction (string name)

  Returns True if the function exists in the current program's function name space.
- bool Qore::existsFunction (code c)

  Always returns True.
- nothing Qore::existsFunction ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ string Qore::functionType (string name)

  Returns `"builtin"` (for a builtin function), `"user"` (for a user function), or NOTHING (if the function cannot be found) according to the function name passed.
- nothing Qore::functionType ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ int Qore::getByte (string str, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- nothing Qore::getByte ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ int Qore::getByte (binary b, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- string Qore::getClassName (object obj)

  Returns the class name of the object passed.

- nothing Qore::getClassName ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::getFeatureList ()

  Returns a list of strings of the builtin and module-supplied features of Qore.

- hash Qore::getModuleHash ()

  Returns a hash of hashes describing the currently-loaded Qore modules; the top-level hash keys are the module names.

- list Qore::getModuleList ()

  Returns a list of hashes describing the currently-loaded Qore modules.

- __7_ int Qore::getWord32 (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int Qore::getWord32 (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- nothing Qore::getWord32 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ int Qore::get_byte (string str, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int Qore::get_byte (binary b, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- string Qore::get_default_encoding ()

  Returns the name of the default character encoding.

- string Qore::get_ex_pos (hash ex)

  returns a descriptive string for an exception location; the `source` and `offset` information will also be included in the string returned if present in the exception hash argument

- int Qore::get_parse_options ()

  returns the current parse options for the current Program object

- hash Qore::get_qore_library_info ()

  Returns a hash of library build and version info.

- hash Qore::get_qore_option_hash ()

  Returns a hash of hashes giving information about Qore library options for the current build.

- list Qore::get_qore_option_list ()

  Returns a list of hashes giving information about Qore library options for the current build.

- __7_ string Qore::get_script_dir ()

  Returns the name of the directory from which the current script was executed or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ string Qore::get_script_name ()

  Returns the filename of the current script if known or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ string Qore::get_script_path ()

Returns the path (directory and filename) of the current script or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ int Qore::get_word_16 (string str, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_16 (binary b, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_16_lsb (string str, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_16_lsb (binary b, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_32 (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_32 (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_32_lsb (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_32_lsb (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_64 (string str, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_64 (binary b, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_64_lsb (string str, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int Qore::get_word_64_lsb (binary b, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- bool Qore::has_key (hash h, string key)

  Returns True if the given key exists in the hash (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered.
- bool Qore::has_key (object obj, string key)

  Returns True if the given key exists in the object (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered or in case of object access errors.
- list Qore::hash_values (hash h)

  Returns a list of all the values in the hash argument passed.
- nothing Qore::hash_values ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::hextoint (string str)

  Returns an integer for a hexadecimal string value; throws an exception if non-hex digits are found.
- nothing Qore::hextoint ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::html_decode (string str)

  Returns a string with any HTML escape codes translated to the original characters.
- nothing Qore::html_decode ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::html_encode (string str)

  Returns a string with characters needing HTML escaping translated to HTML escape codes.
- nothing Qore::html_encode ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing Qore::load_module (string name)

  Loads in a Qore module at run-time.
- nothing Qore::load_module ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::makeBase64String (string str, softint maxlinelen=-1)

  Returns a base64-encoded representation of a string.
- string Qore::makeBase64String (binary bin, softint maxlinelen=-1)

  Returns a base64-encoded representation of a binary object.
- nothing Qore::makeBase64String ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::makeHexString (string str)

  Returns a hex-encoded representation of a string.
- string Qore::makeHexString (binary bin)

  Returns a hex-encoded representation of a binary object.
- nothing Qore::makeHexString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::parse (string code, string label, __7_ softint warning_mask, __7_ string source, __7_ softint offset, softbool format_label=True)

  Adds the text passed to the current program's code, tagged with the given label.
- nothing Qore::parse ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::parseBase64String (string str)

  Parses a base64 encoded string and returns a binary object of the decoded data.
- nothing Qore::parseBase64String ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::parseBase64StringToString (string str, __7_ string encoding)

  Parses a base64 encoded string and returns a string of the decoded data.
- nothing Qore::parseBase64StringToString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary Qore::parseHexString (string hexstr)

  Parses a hex-encoded string and returns the binary object.
- nothing Qore::parseHexString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::parseURL (string url, bool keep_brackets=False)

  Parses a URL string and returns a hash of the components; if the URL cannot be parsed then NOTHING is returned.
- nothing Qore::parseURL ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- hash Qore::parse_url (string url, bool keep_brackets=False)

  Parses a URL string and returns a hash of the components; throws an exception if the string cannot be parsed as a URL.

- string Qore::splice (string str)

  This function always returns an empty string "".

- string Qore::splice (string str, softint start)

  Returns a string based on the argument string but with characters removed from a certain character index.

- string Qore::splice (string str, softint start, softint len, __7_ string nstr)

  Returns a string based on the argument string but optionally with characters removed and/or added from a certain character index.

- list Qore::splice (list l, softint start)

  Returns a list based on the argument list but with elements removed from the given index to the end of the list.

- list Qore::splice (list l, softint start, softint len, __7_ softlist nlist)

  Returns a list based on the argument list but optionally with elements removed and/or added from a certain index.

- nothing Qore::splice ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::strtoint (string num, softint base=10)

  parses a string representing a number in a configurable base and returns the integer

- nothing Qore::strtoint ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.52.1 Detailed Description

Miscellaneous functions

43.52.2 Function Documentation

43.52.2.1 string Qore::backquote (string *cmd*, __7_ reference *rc*)

Executes a process and returns a string of the output (stdout only)

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Parameters

cmd	The shell command to executed as a subprocess
rc	an optional reference to an integer to return the return code of the process

Returns

The stdout of the shell command that was executed

Example:

```
my int $rc;
my string $files = $files = backquote("ls", \$rc);
```

Exceptions

BACKQUOTE-ERROR	An error occurred with the fork() or opening the stdout pipe

See also

- system()
- the backquote operator

Since

Qore 0.8.8 the *rc* argument was added

43.52.2.2 nothing Qore::backquote ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_PROCESS

Code Flags:

RUNTIME_NOOP

43.52.2.3 any Qore::call_builtin_function (string *name*, ...)

Calls a function and returns the return value, passing the remaining arguments after the function name to the builtin function.

Parameters

name	The name of the builtin function to call
...	Any optional arguments to the function

Returns

The value returned by the called function

Example:

```
my any $result = call_builtin_function("func_name", $arg1, $arg2);
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow access to the function
NO-FUNCTION	The function does not exist

Note

The function called could also throw other exceptions

Since

0.8.4 this function no longer restricts its search to builtin functions as as of Qore 0.8.4 builtin and user functions are stored identically internally; there is only one function implementation which can contain both builtin and user variants

43.52.2.4 any Qore::call_builtin_function_args (string *name,* __7_ softlist *vargs*)

Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.

Parameters

name	The name of the builtin function to call
vargs	Optionally a single argument to the function or a list of arguments to the function

Returns

The value returned by the called function

Example:

```
my any $result = call_builtin_function_args("func_name", ($arg1, $arg2));
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow access to the function
NO-FUNCTION	The function does not exist

Note

The function called could also throw other exceptions

Since

0.8.4 this function no longer restricts its search to builtin functions as as of Qore 0.8.4 builtin and user functions are stored identically internally; there is only one function implementation which can contain both builtin and user variants

43.52.2.5 any Qore::call_function (string *name*, ...)

Calls a function and returns the return value, passing the remaining arguments after the function name to the function.

Parameters

name	The name of the function to call
...	Any optional arguments to the function

Returns

The value returned by the called function

Example:

```
my any $result = call_function("func_name", $arg1, $arg2);
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow access to the function
NO-FUNCTION	The function does not exist

Note

The function called could also throw other exceptions

43.52.2.6 any Qore::call_function (code *f*, ...)

Calls the given call reference or closure and returns the result, passing the remaining arguments to the call reference or closure.

Parameters

f	The call reference or closure
...	Any optional arguments to the call reference or closure

Example:

```
my any $result = call_function($call_ref, $arg1, $arg2);
```

Note

The call reference or closure called could also throw other exceptions

43.52.2.7 any Qore::call_function_args (string *name*, __7_ softlist *vargs*)

Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.

Parameters

name	The name of the function to call
vargs	Optionally a single argument to the function or a list of arguments to the function

Returns

The value returned by the called function

Example:

```
my any $result = call_function_args("func_name", ($arg1, $arg2));
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow access to the function
NO-FUNCTION	The function does not exist

Note

The function called could also throw other exceptions

43.52.2.8 any Qore::call_function_args (code *f*, __7_ softlist *vargs*)

Calls the given call reference or closure and returns the result, using the optional second argument as a list of arguments to the call reference or closure.

Parameters

f	The call reference or closure
vargs	Optionally a single argument to the call reference or closure or a list of arguments to the call reference or closure

Example:

```
my any $result = call_function_args($call_ref, ($arg1, $arg2));
```

Note

The call reference or closure called could also throw other exceptions

43.52.2.9 string Qore::decode_url (string *url*)

Decodes percent numeric codes in a URL string and returns the decoded string in UTF-8 encoding.

Code Flags:

CONSTANT

Parameters

url	a URL string with percent-encodings to decode

Returns

the URL string in UTF-8 encoding with all percent-encodings decoded

Example:

```
my string $decoded_url = decode_url($encoded_url);
```

Note

percent decoding is made according to RFC 3986

See also

encode_url()

43.52.2.10 nothing Qore::decode_url ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.11 string Qore::encode_url (string *url*, softbool *encode_all* = False)

Encodes URLs by substituting '%' characters with '%25', spaces (' ') with '%20', and non-ascii characters by percent-encoded representations.

Code Flags:

CONSTANT

Parameters

url	a URL string to encode
encode_all	if True, then in addition to '%', spaces (' '), and non-ascii characters the following reserved characters (according to RFC 3986 are also encoded: '!', '*', '\'', '(', ')', ';', ':', '@', '&', '=', '+', '$', ',', '/', '?', '#', '[', ']'

Returns

the URL string with encoded according to the arguments

Example:

```
my string $encoded_url = encode_url($url);
```

See also

decode_url()

Since

Qore 0.8.9

43.52.2.12 bool Qore::exists (...)

A function performing the same role as the exists operator.

Code Flags:

CONSTANT

Parameters

...	if only a single argument is passed, then this function returns True if the single argument exists, False if not; otherwise is multiple arguments are passed to the function, it always returns True; this is to emulate the behavior of the exists operator

Returns

if only a single argument is passed, then this function returns True if the single argument exists, False if not; otherwise is multiple arguments are passed to the function, it always returns True; this is to emulate the behavior of the exists operator

Example:

```
my bool $b = exists($val);
```

43.52.2.13 bool Qore::existsFunction (string *name*)

Returns True if the function exists in the current program's function name space.

Code Flags:

CONSTANT

Parameters

name	the name of the function to check

Returns

True if the function exists in the current program's function name space, False if not

Example:

```
my bool $b = existsFunction("func_name");
```

43.52.2.14 bool Qore::existsFunction (code *c*)

Always returns True.

This function variant is included for backwards-compatibility

Code Flags:

NOOP

Parameters

c	a call reference or closure:

Returns

always returns True

43.52.2.15 nothing Qore::existsFunction ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.16 __7_ string Qore::functionType (string *name*)

Returns `"builtin"` (for a builtin function), `"user"` (for a user function), or NOTHING (if the function cannot be found) according to the function name passed.

Parameters

name	The function name to check

Returns

`"builtin"` (for a builtin function), `"user"` (for a user function), or NOTHING (if the function cannot be found) according to the function name passed

Example:

```
my *string $type = functionType("print");
```

43.52.2.17 nothing Qore::functionType ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.18 __7_ int Qore::get_byte (string *str,* softint *offset* = 0)

Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the byte offset of the data to retrieve (the first value is at offset 0)

Returns

the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $byte = get_byte($str, 2); # returns the third byte of the string
```

Note

that for strings this function is not equivalent to the [] operator for multi-byte character encodings, as this function works purely on bytes and the [] operator operates on characters

Furthermore this function is identical to getByte(), except this function has no RUNTIME_NOOP variant

43.52.2.19 __7_ int Qore::get_byte (binary *b,* softint *offset* = 0)

Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the byte offset of the data to retrieve (the first value is at offset 0)

Returns

the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $byte = get_byte($bin, 2); # returns the third byte of the
      binary object
```

Note

this function is equivalent to the more efficient [] operator when used with a binary argument

Furthermore this function is identical to getByte(), except this function has no RUNTIME_NOOP variant

43.52.2.20 string Qore::get_default_encoding ( )

Returns the name of the default character encoding.

Returns

the name of the default character encoding

Example:

```
my string $encoding = get_default_encoding();
```

43.52.2.21 string Qore::get_ex_pos (hash *ex*)

returns a descriptive string for an exception location; the `source` and `offset` information will also be included in the string returned if present in the exception hash argument

Code Flags:

CONSTANT

Example:

```
try {
    throw "oops", sprintf("arg %y is invalid", $arg);
}
catch (hash $ex) {
    log("%s: %s: %s", get_ex_pos($ex), $ex.err, $ex.desc);
}
```

Parameters

ex	an exception hash

Returns

a string describing the exception location (ex: `"sftp-poller.q:140"`)

Since

Qore 0.8.7

43.52.2.22 int Qore::get_parse_options ( )

returns the current parse options for the current Program object

Code Flags:

CONSTANT

Example:

```
my int $i = get_parse_options();
```

Returns

the current parse options for the current Program object

Since

Qore 0.8.7

43.52.2.23 hash Qore::get_qore_library_info ( )

Returns a hash of library build and version info.

Returns

a hash of library build and version info with the following keys:

- `PlatformOS`: The operating system used to build the Qore library
- `PlatformCPU`: The CPU used as a target for the Qore library build
- `VersionString`: The full version string for this version of the Qore library
- `VersionMajor`: An integer giving the Qore library's major version number
- `VersionMinor`: An integer giving the Qore library's minor version number
- `VersionSub`: An integer giving the Qore library's release version number
- `Build`: An integer giving the Qore library's subversion revision number
- `BuildHost`: A string giving information about the host used to compile the Qore library
- `Compiler`: The compiler used to build the Qore library
- `ModuleDir`: The module directory assumed by default in the Qore library
- `CFLAGS`: The compiler flags used to compile the Qore library
- `LDFLAGS`: The linker flags used to link the Qore library

Code Flags:

CONSTANT

Example:

```
my hash $h = get_qore_library_info();
```

43.52.2.24 hash Qore::get_qore_option_hash ( )

Returns a hash of hashes giving information about Qore library options for the current build.

Returns

a hash of hashes giving information about Qore library options for the current build; the hash keys are the constant names and the values are hashes with the following keys:

- `option`: The string description of the option
- `constant`: A string giving the name of the constant that has the boolean value for this option (equal to the hash key name)
- `type`: The type of option
- `value`: The boolean value of the option

Code Flags:

CONSTANT

Example:

```
my hash $h = get_qore_option_hash();
```

See also

get_qore_option_list()

Since

Qore 0.8.4

43.52.2.25 list Qore::get_qore_option_list ( )

Returns a list of hashes giving information about Qore library options for the current build.

Returns

a list of hashes giving information about Qore library options for the current build; each hash will have the following keys:

- `option`: The string description of the option
- `constant`: A string giving the name of the constant that has the boolean value for this option
- `type`: The type of option
- `value`: The boolean value of the option

Code Flags:

CONSTANT

Example:

```
my list $l = get_qore_option_list();
```

See also

get_qore_option_hash()

43.52.2.26 __7_ string Qore::get_script_dir ( )

Returns the name of the directory from which the current script was executed or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Returns

the name of the directory from which the current script was executed or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Example:

```
my *string $str = get_script_dir();
```

43.52.2.27 __7_ string Qore::get_script_name ( )

Returns the filename of the current script if known or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Returns

the filename of the current script if known or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Example:

```
my *string $str = get_script_name();
```

43.52.2.28 __7_ string Qore::get_script_path ( )

Returns the path (directory and filename) of the current script or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Returns

the path (directory and filename) of the current script or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

Example:

```
my *string $str = get_script_path();
```

43.52.2.29 __7_ int Qore::get_word_16 (string *str*, softint *offset* = 0)

Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes MSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 2-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_16($str, 2); # returns the third 2-byte (16-bit) value in the
      string
```

See also

get_word_16_lsb()

43.52.2.30 __7_ int Qore::get_word_16 (binary *b*, softint *offset* = 0)

Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes MSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 2-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_16($bin, 2); # returns the third 2-byte (16-bit) value in the
      binary object
```

See also

get_word_16_lsb()

43.52.2.31 __7_ int Qore::get_word_16_lsb (string *str*, softint *offset* = 0)

Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes LSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 2-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_16_lsb($str, 2); # returns the third 2-byte (16-bit) value in the
      string
```

See also

get_word_16()

43.52.2.32 __7_ int Qore::get_word_16_lsb (binary *b*, softint *offset* = 0)

Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes LSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 2-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_16_lsb($bin, 2); # returns the third 2-byte (16-bit) value in the
      binary object
```

See also

get_word_16()

43.52.2.33 __7_ int Qore::get_word_32 (string *str*, softint *offset* = 0)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes MSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_32($str, 4); # returns the third 4-byte (32-bit) value in the
      string
```

Note

This function is identical to getWord32(), except this function has no RUNTIME_NOOP variant

See also

get_word_32_lsb()

43.52.2.34 __7_ int Qore::get_word_32 (binary *b*, softint *offset* = 0)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes MSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_32($bin, 2); # returns the third 4-byte (32-bit) value in the
      binary object
```

Note

This function is identical to getWord32(), except this function has no RUNTIME_NOOP variant

See also

get_word_32_lsb()

43.52.2.35 __7_ int Qore::get_word_32_lsb (string *str,* softint *offset* = 0)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes LSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_32_lsb($str, 4); # returns the third 4-byte (32-bit) value in the
      string
```

Note

This function is identical to getWord32(), except this function has no RUNTIME_NOOP variant

See also

get_word_32()

43.52.2.36 __7_ int Qore::get_word_32_lsb (binary *b*, softint *offset* = 0)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes LSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_32_lsb($bin, 2); # returns the third 4-byte (32-bit) value in the
      binary object
```

See also

get_word_32()

43.52.2.37 __7_ int Qore::get_word_64 (string *str*, softint *offset* = 0)

Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes MSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 8-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_64($str, 4); # returns the third 8-byte (64-bit) value in the
      string
```

See also

get_word_64_lsb()

43.52.2.38 __7_ int Qore::get_word_64 (binary *b*, softint *offset* = 0)

Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes `MSB byte order` when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 8-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_64($bin, 2); # returns the third 8-byte (64-bit) value in the
      binary object
```

See also

get_word_64_lsb()

43.52.2.39 __7_ int Qore::get_word_64_lsb (string *str*, softint *offset* = 0)

Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes `LSB byte order` when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

str	the string data to process
offset	the 8-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_64_lsb($str, 4); # returns the third 8-byte (64-bit) value in the
      string
```

See also

get_word_64()

43.52.2.40 __7_ int Qore::get_word_64_lsb (binary *b*, softint *offset* = 0)

Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes LSB byte order when retrieving the value from the data

Code Flags:

CONSTANT

Parameters

b	the data to process
offset	the 8-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = get_word_64_lsb($bin, 2); # returns the third 8-byte (64-bit) value in the
      binary object
```

See also

get_word_64()

43.52.2.41 __7_ int Qore::getByte (string *str*, softint *offset* = 0)

Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

Code Flags:

CONSTANT, DEPRECATED

Parameters

str	the string data to process
offset	the byte offset of the data to retrieve (the first value is at offset 0)

Returns

the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $byte = get_byte($str, 2); # returns the third byte of the string
```

Note

that for strings this function is not equivalent to the [] operator for multi-byte character encodings, as this function works purely on bytes and the [] operator operates on characters

Furthermore this function is identical to get_byte(), except this function has a RUNTIME_NOOP variant

Since

Qore 0.8.4 marked as deprecated; use the get_byte() function instead

43.52.2.42 nothing Qore::getByte ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP, DEPRECATED

43.52.2.43 __7_ int Qore::getByte (binary *b*, softint *offset* = 0)

Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

Code Flags:

CONSTANT, DEPRECATED

Parameters

b	the data to process
offset	the byte offset of the data to retrieve (the first value is at offset 0)

Returns

the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $byte = get_byte($bin, 2); # returns the third byte of the
      binary object
```

Note

this function is equivalent to the more efficient [] operator when used with a binary argument

Furthermore this function is identical to get_byte(), except this function has a RUNTIME_NOOP variant

Since

Qore 0.8.4 marked as deprecated; use the get_byte() function instead

43.52.2.44 string Qore::getClassName (object *obj*)

Returns the class name of the object passed.

Parameters

obj	the object to get the class name of

Returns

the class name of the object passed

Example:

```
my string $name = getClassName($obj);
```

43.52.2.45 nothing Qore::getClassName ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.46 list Qore::getFeatureList ( )

Returns a list of strings of the builtin and module-supplied features of Qore.

Returns

a list of strings of the builtin and module-supplied features of Qore

Code Flags:

CONSTANT

Example:

```
my list $list = getFeatureList();
```

43.52.2.47 hash Qore::getModuleHash ( )

Returns a hash of hashes describing the currently-loaded Qore modules; the top-level hash keys are the module names.

Returns

a hash of hashes describing the currently-loaded Qore modules; the top-level hash keys are the module names; the values are hashes with the following keys:

- `filename`: the path to the module
- `name`: the name of the module
- `desc`: the description of the module
- `version`: the version of the module
- `author`: the author of the module
- `api_major`: the major number of the Qore module API version the module support
- `api_minor`: the minor number of the Qore module API version the module support
- `url`: the module's URL
- `license`: the module's license

Code Flags:

CONSTANT

Example:

```
my hash $mh = getModuleHash();
```

Since

Qore 0.8.1

43.52.2.48 list Qore::getModuleList ( )

Returns a list of hashes describing the currently-loaded Qore modules.

Returns

a list of hashes describing the currently-loaded Qore modules; each element in the list is a hash with the following keys:

- `filename`: the path to the module
- `name`: the name of the module
- `desc`: the description of the module
- `version`: the version of the module
- `author`: the author of the module
- `api_major`: the major number of the Qore module API version the module support
- `api_minor`: the minor number of the Qore module API version the module support
- `url`: the module's URL
- `license`: the module's license

Code Flags:

CONSTANT

Example:

```
my list $list = getModuleList();
```

See also

getModuleHash()

43.52.2.49 __7_ int Qore::getWord32 (string *str*, softint *offset* = `0`)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes `MSB byte order` when retrieving the value from the data

Code Flags:

CONSTANT, DEPRECATED

Parameters

str	the string data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = getWord32($str, 4); # returns the third 4-byte (32-bit) value in the
      string
```

Note

This function is identical to get_word_32(), except this function has a RUNTIME_NOOP variant

Since

Qore 0.8.4 marked as deprecated; use the get_word_32() function instead

43.52.2.50 __7_ int Qore::getWord32 (binary *b*, softint *offset* = 0)

Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

This function assumes `MSB byte order` when retrieving the value from the data

Code Flags:

CONSTANT, DEPRECATED

Parameters

b	the data to process
offset	the 4-byte offset of the data to retrieve (the first value is at offset 0)

Returns

the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data

Example:

```
my *int $val = getWord32($bin, 2); # returns the third 4-byte (32-bit) value in the
      binary object
```

Note

This function is identical to get_word_32(), except this function has a RUNTIME_NOOP variant

Since

Qore 0.8.4 marked as deprecated; use the get_word_32() function instead

43.52.2.51 nothing Qore::getWord32 ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.52 bool Qore::has_key (hash *h*, string *key*)

Returns True if the given key exists in the hash (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered.

Parameters

h	the hash to check
key	the key name to check; this value will be converted to the default character encoding to check the hash

Returns

True if the given key exists in the hash (does not necessarily have to have a value assigned), False if the key is not present at all

Example:

```
my bool $b = has_key($hash, "key");
```

Exceptions

ENCODING-CONVERSION-ERROR	this error is thrown if the given key cannot be converted to the default character encoding

43.52.2.53 bool Qore::has_key (object *obj*, string *key*)

Returns True if the given key exists in the object (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered or in case of object access errors.

Example:

```
my bool $b = has_key($obj, "key");
```

Exceptions

ENCODING-CONVERSION-ERROR	this error is thrown if the given key cannot be converted to the default character encoding

Note

object access exceptions could also be raised

43.52.2.54 list Qore::hash_values (hash *h*)

Returns a list of all the values in the hash argument passed.

Code Flags:

CONSTANT

Parameters

h	a hash to get all the values of

Returns

a list of all the values in the hash argument passed

Example:

```
my list $vl = hash_values($vl);
```

Note

equivalent to Qore::zzz8hashzzz9::values()

See also

Qore::zzz8hashzzz9::keys()

43.52.2.55 nothing Qore::hash_values ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.56 int Qore::hextoint (string *str*)

Returns an integer for a hexadecimal string value; throws an exception if non-hex digits are found.

Parameters

str	a string of hexadecimal digits (like "6d4f84e0"; with or without leading "x" or "0x")

Returns

the base-10 integer corresponding to the argument

Example:

```
my int $i = hextoint("ab3d4e0f12");
```

Exceptions

PARSE-HEX-ERROR	invalid hex digit found

43.52.2.57 nothing Qore::hextoint ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.58 string Qore::html_decode (string *str*)

Returns a string with any HTML escape codes translated to the original characters.

Code Flags:

CONSTANT

Parameters

str	the argument to decode

Returns

the string passed as an argument with any HTML escape codes translated to the original characters

Example:

```
my string $str = html_decode("&lt;hello&gt;"); # returns "<hello>"
```

43.52.2.59 nothing Qore::html_decode ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.60 string Qore::html_encode (string *str*)

Returns a string with characters needing HTML escaping translated to HTML escape codes.

Code Flags:

CONSTANT

Parameters

str	the argument to process

Returns

the string passed as an argument with characters needing HTML escaping translated to HTML escape codes

Example:

```
my string $str = html_encode("<hello>"); # returns "&lt;hello&gt;"
```

43.52.2.61 nothing Qore::html_encode ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.62 nothing Qore::load_module (string *name*)

Loads in a Qore module at run-time.

If a feature with the same name already exists, then this feature's code is imported into the current Program object if necessary and no further action is taken.

Note that modules providing objects resolved at parse time (classes, constants, functions, etc) must be loaded with the %requires directive instead, unless all references to the objects provided by the module will be made in code embedded in child Program objects.

Restrictions:

Qore::PO_NO_MODULES

Parameters

name	either a feature name (a module will be searched with this feature name) or a path to a module to load

Example:

```
load_module("mysql");
```

Exceptions

LOAD-MODULE-ERROR	module cannot be loaded: API incompatibility, module defines symbols already defined in the target object, etc

See also

getModuleHash(), getFeatureList()

Since

Qore 0.8.7 this function can also be used in user module code

43.52.2.63 nothing Qore::load_module ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_MODULES

Code Flags:

RUNTIME_NOOP

43.52.2.64 string Qore::makeBase64String (string *str,* softint *maxlinelen* = -1)

Returns a base64-encoded representation of a string.

Implementation based on RFC-1421 and RFC-2045

Code Flags:

CONSTANT

Parameters

str	the string to encode
maxlinelen	the maximum length of a line in the resulting output string in bytes; if this value is > 0 then output lines will be separated by CRLF characters

Returns

the base64-encoded string of the data passed

Example:

```
my string $base64 = makeBase64String($data, 64);
```

Since

the maxlinelen parameter was added in Qore 0.8.4

See also

- Qore::zzz8stringzzz9::toBase64()
- makeHexString(string)

43.52.2.65 string Qore::makeBase64String (binary *bin*, softint *maxlinelen* = -1)

Returns a base64-encoded representation of a binary object.

Implementation based on RFC-1421 and RFC-2045

Code Flags:

CONSTANT

Parameters

bin	the data to encode
maxlinelen	the maximum length of a line in the resulting output string in bytes; if this value is > 0 then output lines will be separated by CRLF characters

Returns

the base64-encoded string of the data passed

Example:

```
my string $base64 = makeBase64String($data, 64);
```

Since

the maxlinelen parameter was added in Qore 0.8.4

See also

- Qore::zzz8binaryzzz9::toBase64()
- makeHexString(binary)

43.52.2.66 nothing Qore::makeBase64String ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.67 string Qore::makeHexString (string *str*)

Returns a hex-encoded representation of a string.

Code Flags:

CONSTANT

Parameters

str	the string to encode

Returns

a string of hex digits repsenting the bytes of the argument passed

Example:

```
my string $str = makeHexString($str2);
```

See also

- Qore::zzz8stringzzz9::toHex()
- makeBase64String(string)
- parseHexString()

43.52.2.68 string Qore::makeHexString (binary *bin*)

Returns a hex-encoded representation of a binary object.

Parameters

bin	the binary object to encode

Returns

a string of hex digits repsenting the bytes of the argument passed

Example:

```
my string $str = makeHexString($bin);
```

See also

- Qore::zzz8binaryzzz9::toHex()
- makeBase64String(binary)
- parseHexString()

43.52.2.69 nothing Qore::makeHexString ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.70 __7_ hash Qore::parse (string *code*, string *label*, __7_ softint *warning_mask*, __7_ string *source*, __7_ softint *offset*, softbool *format_label* = `True`)

Adds the text passed to the current program's code, tagged with the given label.

Restrictions:

Qore::PO_NO_EMBEDDED_LOGIC, Qore::PO_IN_MODULE

Parameters

code	the string of Qore source code to parse; the parsed code will be added to the current program
label	a label identifying the code or the code's source
warning_mask	An optional warning mask; see Warning Constants for values to combine by binary-or; if this arguments is 0 or not given then no warnings will be checked or issued and the return value will always be NOTHING
source	An optional source file name for the code being parsed; this is useful if sections of a file are parsed
offset	An optional line offset for use with the *source* parameter; this gives the line offset in the file to the code being parsed
format_label	If this argument is True, then the label is formatted as "`<run-time-loaded : label>`"; if False, then it is passed as-is

Example:

```
parse($code, $filename);
```

Note

This function could throw many parse exceptions which are not enumerated here; any parse errors will result in an appropriate exception.

See also

- Qore::Program::parse()
- Qore::Program::parsePending()

Since

Qore 0.8.7 the optional *warning_mask*, *source*, *offset*, and *format_label* arguments were added; also label formatting was added; these changes were made to align the functionality of this function with the Program::parse() and Program::parsePending() methods

43.52.2.71 nothing Qore::parse ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EMBEDDED_LOGIC, Qore::PO_IN_MODULE

Code Flags:

RUNTIME_NOOP

43.52.2.72 hash Qore::parse_url (string *url*, bool *keep_brackets* = `False`)

Parses a URL string and returns a hash of the components; throws an exception if the string cannot be parsed as a URL.

Parameters

url	the URL to parse (ex: `"https://user:pass@host:8080/path"`); either a hostname or path is required at a minimum or a `PARSE-URL-ERROR` exception is raised
keep_brackets	if this argument is true then if the hostname or address is enclosed in square brackets, then the brackets will be included in the `"host"` key output as well; square brackets are used by some Qore methods to denote IPv6 addresses; for example see Socket::connect()

Returns

a hash of the components of the URL with the following keys (if data in the URL is present; note that at least either the `"host"` or the `"path"` keys will always be returned if no `PARSE-URL-ERROR` is raised):

- `protocol`: the scheme in the URL (ex: `"http"`)
- `path`: any path given in the URL; the path will be prefixed by `"/"` if a hostname is found in the URL argument string, otherwise it will not if it was not given as such in the argument string
- `username`: any username given in the URL
- `password`: any password given in the URL
- `host`: any hostname given in the URL; note that this key will be given if no other information can be found in the URL argument and the URL argument string has no `"/"` characters; depending on the usage context for this function, this may actually be a filename
- `port`: any port number given in the URL

Example:

```
my hash $hash = parse_url($url_string);
```

Exceptions

PARSE-URL-ERROR	The URL string given could not be parsed

Note

URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme:///socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: `"http://socket=%2ftmp%socket-dir%2fsocket-file-1/` this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

See also

parseURL() for a version of this function that does not throw exceptions if the URL cannot be parsed

43.52.2.73 binary Qore::parseBase64String (string *str*)

Parses a base64 encoded string and returns a binary object of the decoded data.

Implementation based on RFC-1421 and RFC-2045

Parameters

str	the base64-encoded input data to decode

Returns

a binary object of the decoded data

Example:

```
my binary $bin = parseBase64String($base64_string);
```

Exceptions

BASE64-PARSE-ERROR	A syntax error occurred parsing the base64 string (invalid character, etc)

See also

parseHexString(), parseBase64StringToString()

43.52.2.74 nothing Qore::parseBase64String ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.75 string Qore::parseBase64StringToString (string *str*, __7_ string *encoding*)

Parses a base64 encoded string and returns a string of the decoded data.

Implementation based on RFC-1421 and RFC-2045

Parameters

str	the base64-encoded input data to decode
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed.

Returns

a string of the decoded data tagged with the given encoding

Example:

```
my string $str = parseBase64String($base64_string);
```

Exceptions

BASE64-PARSE-ERROR	A syntax error occurred parsing the base64 string (invalid character, etc)

Since

the encoding parameter was added in Qore 0.8.4

See also

parseBase64String()

43.52.2.76 nothing Qore::parseBase64StringToString ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.77 binary Qore::parseHexString (string *hexstr*)

Parses a hex-encoded string and returns the binary object.

Parameters

hexstr	a string of an even-number of only hexadecimal digits

Returns

a binary object of the decoded data

Example:

```
my binary $bin = parseHexString($hex_string);
```

Exceptions

PARSE-HEX-ERROR	A syntax error occurred parsing the hex string (odd number of digits, invalid hex character, etc)

See also

parseBase64String()
makeHexString();

43.52.2.78 nothing Qore::parseHexString ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.79 __7_ hash Qore::parseURL (string *url*, bool *keep_brackets* = `False`)

Parses a URL string and returns a hash of the components; if the URL cannot be parsed then NOTHING is returned.

Parameters

url	the URL to parse (ex: `"https://user:pass@host:8080/path"`); either a hostname or path is required at a minimum or the function will return NOTHING
keep_brackets	if this argument is true then if the hostname or address is enclosed in square brackets, then the brackets will be included in the `"host"` key output as well; square brackets are used by some Qore methods to denote IPv6 addresses; for example see Socket::connect()

Returns

a hash of the components of the URL with the following keys (if data in the URL is present; note that at least either the `"host"` or the `"path"` keys will always be returned if a hash is returned):

- `protocol`: the scheme in the URL (ex: `"http"`)
- `path`: any path given in the URL; the path will be prefixed by `"/"` if a hostname is found in the URL argument string, otherwise it will not if it was not given as such in the argument string
- `username`: any username given in the URL
- `password`: any password given in the URL
- `host`: any hostname given in the URL; note that this key will be given if no other information can be found in the URL argument and the URL argument string has no `"/"` characters; depending on the usage context for this function, this may actually be a filename
- `port`: any port number given in the URL

Example:

```
my *hash $hash = parseURL($url_string);
```

Note

URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme://socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: `"http://socket=%2ftmp%socket-dir%2fsocket-file-1/` this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

See also

parse_url() for a version of this function that throws exceptions if the URL cannot be parsed

43.52.2.80 nothing Qore::parseURL ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.81 string Qore::splice (string *str*)

This function always returns an empty string `""`.

This function variant is included for backwards-compatibility

Code Flags:

NOOP

Parameters

str	no action is taken on the argument

Returns

always returns an empty string " "

43.52.2.82 string Qore::splice (string *str*, softint *start*)

Returns a string based on the argument string but with characters removed from a certain character index.

An exception can only be thrown here if an invalid multi-byte encoding is found

Parameters

str	the string to process
start	the character index (where the first character is 0) to start removing characters; if this value is negative, it gives the offset from the end of the string

Returns

the processed string

Example:

```
my string $str = splice($str2, 5);
```

See also

the splice operator

43.52.2.83 string Qore::splice (string *str*, softint *start*, softint *len*, __7_ string *nstr*)

Returns a string based on the argument string but optionally with characters removed and/or added from a certain character index.

An exception can only be thrown here if an invalid multi-byte encoding is found

Parameters

str	the string to process
start	the character index (where the first character is 0) to start removing (and/or adding) characters; if this value is negative, it gives the offset from the end of the string
len	the number of characters to remove; if this argument is 0 then no characters are removed; if this value is negative, then it gives an offset from the end of the string (ie -2 means remove all characters up to but not including the two last characters)
nstr	the optional string for inserting new characters

Returns

the processed string with characters removed and/or added according to the arguments

Example:

```
my string $str = splice($str2, 5, 7, "hello");
```

See also

the splice operator

43.52.2.84 list Qore::splice (list *l*, softint *start*)

Returns a list based on the argument list but with elements removed from the given index to the end of the list.

Exceptions can only be thrown here if objects are removed from the list and this causes them to go out of scope and an exception is thrown in one of the destructors

Parameters

l	the list to process
start	the starting element (where the first element is 0) to start removing elements; if this value is negative, it gives the offset from the end of the list

Example:

```
my list $l = splice($l2, 5);
```

See also

the splice operator

43.52.2.85 list Qore::splice (list *l*, softint *start*, softint *len*, __7_ softlist *nlist*)

Returns a list based on the argument list but optionally with elements removed and/or added from a certain index.

Exceptions can only be thrown here if objects are removed from the list and this causes them to go out of scope and an exception is thrown in one of the destructors

Parameters

l	the list to process
start	the starting element (where the first element is 0) to start removing (and/or adding) elements; if this value is negative, it gives the offset from the end of the list
len	the number of list elements to remove; if this argument is 0 then no elements are removed; if this value is negative, then it gives an offset from the end of the list (ie -2 means remove all elements up to but not including the two last elements)
nlist	the optional list for inserting new elements

Returns

the processed list with elements removed and/or added according to the arguments

Example:

```
my list $l = splice($l2, 5, 7, "hello");
```

See also

the splice operator

43.52.2.86 nothing Qore::splice ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.52.2.87 int Qore::strtoint (string *num*, softint *base* = 10)

parses a string representing a number in a configurable base and returns the integer

Parameters

num	a string representing a number
base	the base of the number

Returns

the integer represented by the string and the base

Example:

```
my int $i = strtoint("41", 8);
```

Exceptions

STRTOINT-ERROR	cannot parse string; unsupported base, etc

43.52.2.88 nothing Qore::strtoint ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.53 Signal Constants

Variables

- const Qore::NameToSignal

  maps signal names to signal values
- const Qore::SIGABRT = SIGABRT

  SIGABRT.
- const Qore::SIGALRM = SIGALRM

  SIGALRM.
- const Qore::SIGBUS = SIGBUS

  SIGBUS.
- const Qore::SIGCANCEL = SIGCANCEL

  SIGCANCEL.
- const Qore::SIGCHLD = SIGCHLD

  SIGCHLD.
- const Qore::SIGCLD = SIGCLD

  SIGCLD.
- const Qore::SIGCONT = SIGCONT

  SIGCONT.
- const Qore::SIGEMT = SIGEMT

  SIGEMT.
- const Qore::SIGFPE = SIGFPE

  SIGFPE.
- const Qore::SIGFREEZE = SIGFREEZE

  SIGFREEZE.
- const Qore::SIGHUP = SIGHUP

  SIGHUP.
- const Qore::SIGILL = SIGILL

  SIGILL.
- const Qore::SIGINFO = SIGINFO

  SIGINFO.
- const Qore::SIGINT = SIGINT

  SIGINT.
- const Qore::SIGIO = SIGIO

  SIGIO.
- const Qore::SIGIOT = SIGIOT

  SIGIOT.
- const Qore::SIGJVM1 = SIGJVM1

  SIGJVM1.
- const Qore::SIGJVM2 = SIGJVM2

  SIGJVM2.
- const Qore::SIGKILL = SIGKILL

  SIGKILL.
- const Qore::SIGLOST = SIGLOST

  SIGLOST.
- const Qore::SIGLWP = SIGLWP

  SIGLWP.
- const Qore::SIGPIPE = SIGPIPE

  SIGPIPE.
- const Qore::SIGPOLL = SIGPOLL

SIGPOLL.

- const Qore::SIGPROF = SIGPROF

  SIGPROF.
- const Qore::SIGPWR = SIGPWR

  SIGPWR.
- const Qore::SIGQUIT = SIGQUIT

  SIGQUIT.
- const Qore::SIGSEGV = SIGSEGV

  SIGSEGV.
- const Qore::SIGSTKFLT = SIGSTKFLT

  SIGSTKFLT.
- const Qore::SIGSTOP = SIGSTOP

  SIGSTOP.
- const Qore::SIGSYS = SIGSYS

  SIGSYS.
- const Qore::SIGTERM = SIGTERM

  SIGTERM.
- const Qore::SIGTHAW = SIGTHAW

  SIGTHAW.
- const Qore::SIGTRAP = SIGTRAP

  SIGTRAP.
- const Qore::SIGTSTP = SIGTSTP

  SIGTSTP.
- const Qore::SIGTTIN = SIGTTIN

  SIGTTIN.
- const Qore::SIGTTOU = SIGTTOU

  SIGTTOU.
- const Qore::SIGURG = SIGURG

  SIGURG.
- const Qore::SIGUSR1 = SIGUSR1

  SIGUSR1.
- const Qore::SIGUSR2 = SIGUSR2

  SIGUSR2.
- const Qore::SIGVTALRM = SIGVTALRM

  SIGVTALRM.
- const Qore::SIGWAITING = SIGWAITING

  SIGWAITING.
- const Qore::SIGWINCH = SIGWINCH

  SIGWINCH.
- const Qore::SIGXCPU = SIGXCPU

  SIGXCPU.
- const Qore::SIGXFSZ = SIGXFSZ

  SIGXFSZ.
- const Qore::SIGXRES = SIGXRES

  SIGXRES.
- const Qore::SignalToName

  maps signal numbers (as a string key) to the symbolic name for the signal

43.53.1 Detailed Description

Signal constants - if any of the constants in this section are not defined on the host; the constant's value will be 0

43.54 Object Functions

Functions

- any Qore::callObjectMethod (object obj, string method,...)

  Calls a method of an object, passing the remainder of the arguments to the function as arguments to the method.
- nothing Qore::callObjectMethod ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- any Qore::callObjectMethodArgs (object obj, string method, __7_ softlist varg)

  Calls a method of an object, using the optional third argument as the argument list to the method.
- nothing Qore::callObjectMethodArgs ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- any Qore::call_pseudo (any val, string meth,...)

  calls a pseudo-method on the given value
- any Qore::call_pseudo_args (any val, string meth, __7_ softlist argv)

  calls a pseudo-method on the given value with arguments given as a list
- list Qore::getMethodList (object obj)

  Returns a list of strings of the names of the methods of the class of the object passed as a parameter.
- nothing Qore::getMethodList ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.54.1 Detailed Description

Object functions

43.54.2 Function Documentation

43.54.2.1 any Qore::call_pseudo (any *val*, string *meth*, ...)

calls a pseudo-method on the given value

Parameters

val	the value to call the pseudo-method on
meth	the string method name of the pseudo-method to call
...	any other arguments to the method

Returns

the return value of the pseudo-method

See also

Qore::zzz8valuezzz9 for the class hierarchy of pseudo-classes

Note

this function can be used for security to ensure that a given pseudo-method of the Qore::zzz8objectzzz9 class is called, as by default if a method of the same name is implemented by the object's class, the class method will be called instead.

Since

Qore 0.8.5

43.54.2.2 any Qore::call_pseudo_args (any *val*, string *meth*, __7_ softlist *argv*)

calls a pseudo-method on the given value with arguments given as a list

Parameters

val	the value to call the pseudo-method on
meth	the string method name of the pseudo-method to call
argv	any other arguments to the method

Returns

the return value of the pseudo-method

See also

Qore::zzz8valuezzz9 for the class hierarchy of pseudo-classes

Note

this function can be used for security to ensure that a given pseudo-method of the Qore::zzz8objectzzz9 class is called, as by default if a method of the same name is implemented by the object's class, the class method will be called instead.

Since

Qore 0.8.8

43.54.2.3 any Qore::callObjectMethod (object *obj*, string *method*, ...)

Calls a method of an object, passing the remainder of the arguments to the function as arguments to the method.

Parameters

obj	the object to use for the call
method	the method to call
...	any additional arguments to the method

Returns

the value returned by the method call

Example:

```
my any $result = callObjectMethod($obj, "method", $arg1, $arg2);
```

Exceptions

METHOD-DOES-NOT-EX↩ IST	The named method does not exist in this class
ILLEGAL-EXPLICIT-MET↩ HOD-CALL	The named method may not be called explicitly
METHOD-IS-PRIVATE	The named method is private and therefore can only be called within the class
BASE-CLASS-IS-PRIVATE	The named method is a member of a privately inherited base class

Note

the method called could cause additional exceptions to be thrown

43.54.2.4 nothing Qore::callObjectMethod ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.54.2.5 any Qore::callObjectMethodArgs (object *obj*, string *method*, ...7_ softlist *varg*)

Calls a method of an object, using the optional third argument as the argument list to the method.

Parameters

obj	the object to use for the call
method	the method to call
varg	any additional arguments to the method

Returns

the value returned by the method call

Example:

```
my any $result = callObjectMethodArgs($obj, "method", $arglist);
```

Exceptions

METHOD-DOES-NOT-EX↩ IST	The named method does not exist in this class
ILLEGAL-EXPLICIT-MET↩ HOD-CALL	The named method may not be called explicitly
METHOD-IS-PRIVATE	The named method is private and therefore can only be called within the class
BASE-CLASS-IS-PRIVATE	The named method is a member of a privately inherited base class

Note

the method called could cause additional exceptions to be thrown

43.54.2.6 nothing Qore::callObjectMethodArgs ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.54.2.7 list Qore::getMethodList (object *obj*)

Returns a list of strings of the names of the methods of the class of the object passed as a parameter.

Code Flags:

CONSTANT

Parameters

obj	an object of the class to get the method list from

Returns

a list of strings of the names of the methods of the class of the object passed as a parameter; returns all methods in the class, both private and public but does not return inherited methods

Example:

```
my list $l = getMethodList($obj);
```

43.54.2.8 nothing Qore::getMethodList ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.55 UNIX User and Group Functions

Functions

- __7_ hash Qore::getgrgid (softint gid)

  Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist NOTHING is returned.
- hash Qore::getgrgid2 (softint gid)

  Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist, a `GETGRGID2-ERROR` exception is thrown.
- __7_ hash Qore::getgrnam (string name)

  Returns a group information hash representing the group information for the group name passed, or, if the group does not exist NOTHING is returned.
- hash Qore::getgrnam2 (string name)

  Returns a group information hash representing the group information for the group name passed, or, if the group does not exist, a `GETGRNAM2-ERROR` exception is thrown.
- hash Qore::getpwnam (string name)

  Returns a password information hash representing the user information for the user name passed, or, if the user does not exist NOTHING is returned.
- hash Qore::getpwnam2 (string name)

  Returns a password information hash representing the user information for the user name passed, or, if the user does not exist, a `GETPWNAM2-ERROR` exception is thrown.
- nothing Qore::getpwuid ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash Qore::getpwuid (softint uid)

  Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist NOTHING is returned.
- hash Qore::getpwuid2 (softint uid)

  Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist, a `GETPWUID2-ERROR` exception is thrown.

43.55.1 Detailed Description

UNIX user and group functions

43.55.2 Group Information Hash

Key	Type	Description
`gr_name`	string	The name of the group
`gr_passwd`	string	The encrypted password for the group
`gr_gid`	int	The group id
`gr_mem`	list	List of strings giving the usernames of members of the group

43.55.3 Password Information Hash

Key	Type	Description

pw_name	string	The username of the user
pw_passwd	string	The encrypted password for the user
pw_gecos	string	The real name or description of the user
pw_dir	string	The user's home directory
pw_shell	string	The user's login shell
pw_uid	int	The user's userid
pw_gid	int	The group id of the user's primary group

43.55.4 Function Documentation

43.55.4.1 __7_ hash Qore::getgrgid (softint *gid*)

Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist NOTHING is returned.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

gid	the group ID to look up

Returns

a group information hash representing the group information for the group ID passed, or, if the group ID does not exist NOTHING is returned

Example:

```
my *hash $hash = getgrgid(0);
```

See also

getgrgid2() for a similar function that throws an exception if the group ID is invalid

43.55.4.2 hash Qore::getgrgid2 (softint *gid*)

Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist, a GETGRGID2-ERROR exception is thrown.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Parameters

gid	the group ID to look up

Returns

a group information hash representing the group information for the group ID passed, or, if the group ID does not exist, a `GETGRGID2-ERROR` exception is thrown

Example:

```
my *hash $hash = getgrgid2(0);
```

Exceptions

GETGRGID2-ERROR	invalid gid or error reading group information

See also

getgrgid() for a similar function that does not throw an exception if the group ID is invalid

43.55.4.3 __7_ hash Qore::getgrnam (string *name*)

Returns a group information hash representing the group information for the group name passed, or, if the group does not exist NOTHING is returned.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

name	the group name to look up

Returns

a group information hash representing the group information for the group name passed, or, if the group does not exist NOTHING is returned

Example:

```
my *hash $hash = getgrnam("root");
```

See also

getgrnam2() for a similar function that throws an exception if the group is invalid

43.55.4.4 hash Qore::getgrnam2 (string *name*)

Returns a group information hash representing the group information for the group name passed, or, if the group does not exist, a GETGRNAM2-ERROR exception is thrown.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Parameters

name	the group name to look up

Returns

a group information hash representing the group information for the group name passed, or, if the group does not exist, a GETGRNAM2-ERROR exception is thrown

Example:

```
my *hash $hash = getgrnam2("root");
```

Exceptions

GETGRNAM2-ERROR	invalid group or error reading group information

See also

getgrnam() for a similar function that does not throw an exception if the group is invalid

43.55.4.5 hash Qore::getpwnam (string *name*)

Returns a password information hash representing the user information for the user name passed, or, if the user does not exist NOTHING is returned.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

name	the user name to look up

Returns

a password information hash representing the user information for the user name passed, or, if the user does not exist NOTHING is returned

Example:

```
my *hash $hash = getpwnam("root");
```

See also

getpwnam2() for a similar function that throws an exception if the user is invalid

43.55.4.6 hash Qore::getpwnam2 (string *name*)

Returns a password information hash representing the user information for the user name passed, or, if the user does not exist, a `GETPWNAM2-ERROR` exception is thrown.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Parameters

name	the user name to look up

Returns

a password information hash representing the user information for the user name passed, or, if the user does not exist, a `GETPWNAM2-ERROR` exception is thrown

Example:

```
my *hash $hash = getpwnam2("root");
```

Exceptions

GETPWNAM2-ERROR	invalid user or error reading user information

See also

getpwnam() for a similar function that does not throw an exception if the user is invalid

43.55.4.7 nothing Qore::getpwuid ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RUNTIME_NOOP

43.55.4.8 __7_ hash Qore::getpwuid (softint *uid*)

Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist NOTHING is returned.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

CONSTANT

Parameters

uid	the user ID to look up

Returns

a password information hash representing the user information for the user ID passed, or, if the user ID does not exist NOTHING is returned

Example:

```
my *hash $hash = getpwuid(0);
```

See also

getpwuid2() for a similar function that throws an exception if the user ID is invalid

43.55.4.9 hash Qore::getpwuid2 (softint *uid*)

Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist, a `GETPWUID2-ERROR` exception is thrown.

Platform Availability:

Qore::Option::HAVE_UNIX_USERMGT

Restrictions:

Qore::PO_NO_EXTERNAL_INFO

Code Flags:

RET_VALUE_ONLY

Parameters

uid	the user ID to look up

Returns

a password information hash representing the user information for the user ID passed, or, if the user ID does not exist, a `GETPWUID2-ERROR` exception is thrown

Example:

```
my *hash $hash = getpwuid2(0);
```

Exceptions

GETPWUID2-ERROR	invalid uid or error reading user information

See also

getpwuid() for a similar function that does not throw an exception if the user ID is invalid

43.56 String Functions

Functions

- int Qore::bindex (softstring str, softstring substr, softint pos=0)

  Retrieves the byte position of a substring within a string.
- int Qore::bindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::brindex (softstring str, softstring substr, softint pos=-1)

  Retrieves the byte position of a substring within a string, starting the search from the end of the string.
- int Qore::brindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::chomp (string str)

  Removes the trailing end-of-line indicator ("\n" or "\r\n") from a string and returns the new string (also see the chomp operator)
- __7_ string Qore::chomp (reference str)

  Removes the trailing end-of-line indicator ("\n" or "\r\n") from a reference to a string and returns the new string (also see the chomp operator)
- nothing Qore::chomp ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::chr (softint val, __7_ string encoding)

  Returns a string containing a single ASCII character represented by the numeric value passed.
- string Qore::chr (any arg)

  This function variant returns a string with a single ASCII NULL ('\0'); it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing Qore::chr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::convert_encoding (string str, string encoding)

  Performs explicit string character encoding conversions.
- nothing Qore::convert_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::f_printf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string; enforces field widths on arguments larger than the given field width.
- string Qore::f_printf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::f_sprintf (string fmt,...)

  Returns a formatted string based on a format string and other arguments; enforces field widths on arguments larger than the given field width.
- string Qore::f_sprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::f_vprintf (string fmt, any varg)

  Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; enforces field widths on arguments larger than the given field width.
- string Qore::f_vsprintf (string fmt, any varg)

Returns a formatted string based on a format string and other arguments given as a list after the format string; enforces field widths on arguments larger than the given field width.

- nothing Qore::flush ()

  Flushes output to the console output with print(), printf(), etc.

- string Qore::force_encoding (string str, string encoding)

  Returns the first string argument tagged with the character encoding given as the second argument; does not actually change the string data.

- nothing Qore::force_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::format_number (string fmt, softfloat num)

  Returns a string of a formatted number according to a number argument and a format string.

- nothing Qore::format_number ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::get_encoding (string str)

  Returns a string describing the character encoding of the string argument passed.

- nothing Qore::get_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::index (softstring str, softstring substr, softint pos=0)

  Retrieves the character position of a substring within a string.

- int Qore::index ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::join (string str,...)

  Creates a string from separator string and a list of arguments.

- string Qore::join (string str, list l)

  Creates a string from separator string and a list of arguments.

- nothing Qore::join ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::length (softstring str)

  Returns the length in characters for the string passed.

- int Qore::length (binary bin)

  Returns the number of bytes in the binary object passed as an argument.

- nothing Qore::length ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::length (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::ord (softstring str, softint offset=0)

  Gives the numeric value of the given byte in the string passed; if no string is passed or the offset is after the end of the string, -1 is returned.

- int Qore::ord ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool Qore::parse_boolean (string str)

  tries to parse a string value as a boolean

- bool Qore::parse_boolean (...)

  returns the first value passed as a boolean

- nothing Qore::print (...)

Outputs a string to standard output with no formatting.

- string Qore::printf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string; does not enforce field widths on arguments larger than the given field width.

- string Qore::printf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool Qore::regex (string str, string regex, int options=0)

  Returns True if the regular expression matches the string passed, otherwise returns False.

- nothing Qore::regex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ list Qore::regex_extract (string str, string regex, int options=0)

  Returns a list of substrings in a string based on matching patterns defined by a regular expression.

- nothing Qore::regex_extract ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::regex_subst (string str, string regex, string subst, int options=0)

  Returns a string with patterns substituted according to the arguments passed.

- nothing Qore::regex_subst ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::replace (string str, string source, string target, int start=0, int end=-1)

  Replaces all occurrences of a substring in a string with another string.

- nothing Qore::replace ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::reverse (softstring str)

  Reverses a string and returns the new string.

- int Qore::rindex (softstring str, softstring substr, softint pos=-1)

  Retrieves the character position of a substring within a string, starting the search from the end of the string.

- int Qore::rindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list Qore::split (string sep, string str, bool with_separator=False)

  Splits a string into a list of components based on a separator string.

- list Qore::split (string sep, string str, string quote, bool trim_unquoted=False)

  Splits a string into a list of components based on a separator string and a quote character.

- list Qore::split (binary sep, binary data)

  Returns a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed.

- list Qore::split ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::sprintf (string fmt,...)

  Returns a formatted string based on a format string and other arguments; does not enforce field widths on arguments larger than the given field width.

- string Qore::sprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::strlen (softstring str)

  Returns the length in bytes of the string argument.

- nothing Qore::strlen ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::strlen (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::strmul (softstring str, softint smul, __7_ softint offset)

  Returns a new string with a repeated string element and optionally removing trailing characters.

- string Qore::substr (softstring str, softint start)

  Returns a portion of a string starting from an integer offset.

- string Qore::substr (softstring str, softint start, softint len)

  Returns a portion of a string starting from an integer offset, with a length parameter.

- binary Qore::substr (binary b, softint start)

  Returns a portion of a binary object starting from an integer offset.

- binary Qore::substr (binary b, softint start, softint len)

  Returns a portion of a binary object starting from an integer offset, with a length parameter.

- nothing Qore::substr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::tolower (string str)

  Returns a string in all lower-case characters based on the argument passed.

- nothing Qore::tolower ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::toupper (string str)

  Returns a string in all upper-case characters based on the argument passed.

- nothing Qore::toupper ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::trim (string str, __7_ string chars)

  Removes byte characters from the start and end of a string and returns the new string (also see the trim operator)

- __7_ string Qore::trim (reference str, __7_ string chars)

  Removes byte characters from the start and end of a reference to an lvalue containing a string and returns string after processing (also see the trim operator)

- nothing Qore::trim ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::trunc_str (softstring str, softint len, __7_ string encoding)

  Returns a truncated string with no more than the given number of bytes and optionally converted to a specific character encoding.

- string Qore::vprintf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; does not enforce field widths on arguments larger than the given field width.

- string Qore::vprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::vsprintf (string fmt, any varg)

  Returns a formatted string based on a format string and other arguments given as a list after the format string; does not enforce field widths on arguments larger than the given field width.

- string Qore::vsprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.56.1 Detailed Description

43.56.2 String Formatting

String formatting in Qore is based on c-style "printf" string formatting.

There are three types of objects in the format string:

- Plain characters, which are copied as-is to the output string
- Escape characters, which are converted and copied to the output string
- Format specifications for arguments to be included in the output string, which are preceded by the percent sign "%"

Note

The percent sign "%" always starts a format specification unless it is followed by another "%" sign, in this case only one "%" is output (ie "%%" in the format string is output as a single "%")

43.56.2.1 format_specification

After the "%" sign, there can be zero or more formatting flags as in the following table:

Printf Formatting Flags

Flag	Description
-	left-justify the field
+	include the sign for the number (+ or -)
0	use zero left padding rather than space padding
space	use space padding

Then a field width specifier can optionally be given as a string of digits specifying the length of the field. With "field" functions (function names preceded by "f_"), these width specifiers are hard limits; that is; arguments longer than the width specified are truncated to the specified width.

For floating-point arguments, a precision specifier may be given by including a period "." and another digit string, which indicates the number of digits to appear after the decimal point.

Then the format character is given as follows:

Printf Formatting Characters

Character	Description
s	string
d	Integer, output in base 10 format
f	Literal floating point value
F	Literat floating point value
a	Hexadecimal floating-point output with lower-case "0x" and "abcdef"
A	Hexadecimal floating-point output with upper-case "0X" and "ABCDEF"
g	compact floating-point output using a lower-case "e" for exponential output when necessary
G	compact floating-point output using an upper-case "E" for exponential output when necessary
n	Any Qore value, formatted as a string, without any line breaks

N	Any Qore value, formatted as a string, with line breaks and whitespace formatting for complex objects
x	Integer, output in base 16 (hexadecimal) format with lower-case a-f
X	Integer, output in base 16 (hexadecimal) format with upper-case A-F
y	Any Qore value, formatted as a pseudo-YAML string, without any line breaks; this format option is similar to "%n", but uses a YAML-like format. Note that the output is not guaranteed to be parsed by the yaml module's parseYAML() function; it is for display purposes only; to get real YAML functionality, use the yaml module.

String Escape Characters

Escape Characters	Description
\n	a newline character
\r	a carriage-return character
\t	a tab character
\b	a backspace character
\f	a form-feed character
num	an 8-bit character whose value is the 1, 2, or 3 digit octal number *num*
\"	a double-quote character (")

43.56.3 Function Documentation

43.56.3.1 int Qore::bindex (softstring *str*, softstring *substr*, softint *pos* = 0)

Retrieves the byte position of a substring within a string.

The *pos* argument and the return value are in byte positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

CONSTANT

Parameters

str	the string to search in
substr	the substring to find in *str*
pos	the starting character position for the search

Returns

the byte position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = bindex($str, $substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Note

no character encoding conversions are made by this function even if the string arguments have different character encodings

See also

- Qore::zzz8stringzzz9::find(softstring, softint)
- Qore::zzz8stringzzz9::rfind(softstring, softint)
- index(softstring, softstring, softint)
- rindex(softstring, softstring, softint)
- brindex(softstring, softstring, softint)

43.56.3.2 int Qore::bindex ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.3 int Qore::brindex (softstring *str*, softstring *substr*, softint *pos* = -1)

Retrieves the byte position of a substring within a string, starting the search from the end of the string.

The *pos* argument and the return value are in byte positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

CONSTANT

Parameters

str	the string to search in
substr	the substring to find in *str*
pos	the starting character position for the search, -1 means start from the end of the string

Returns

the byte position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = brindex($str, $substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Note

no character encoding conversions are made by this function even if the string arguments have different character encodings

See also

- Qore::zzz8stringzzz9::find(softstring, softint)
- Qore::zzz8stringzzz9::rfind(softstring, softint)
- index(softstring, softstring, softint)
- bindex(softstring, softstring, softint)
- rindex(softstring, softstring, softint)

43.56.3.4 int Qore::brindex ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.5 string Qore::chomp (string *str*)

Removes the trailing end-of-line indicator ("\n" or "\r\n") from a string and returns the new string (also see the chomp operator)

If no EOL indicator is present in the string, this function simply returns the original string unmodified.

Code Flags:

CONSTANT

Parameters

str	the string to process

Returns

the new string with any end-of-line character(s) removed

Example:

```
my string $line = chomp("hello\n"); # returns "hello"
```

See also

the chomp operator

43.56.3.6 __7_ string Qore::chomp (reference *str*)

Removes the trailing end-of-line indicator ("\n" or "\r\n") from a reference to a string and returns the new string (also see the chomp operator)

This variant accepts variable references, in which case it will modify the string in place and also return the modified string

Parameters

str	a reference to an lvalue continaing the string value to process; if the lvalue does not have a string value NOTHING is returned

Returns

the string as modified

Example:

```
chomp(\$str);
```

43.56.3.7 nothing Qore::chomp ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.8 string Qore::chr (softint *val*, __7_ string *encoding*)

Returns a string containing a single ASCII character represented by the numeric value passed.

Code Flags:

CONSTANT

Parameters

val	the byte value of the character; only the least-significant byte is used
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string containing a single ASCII character represented by the numeric value passed

Example:

```
my string $char = chr(13);
```

Since

Qore 0.8.5 the encoding argument is supported

43.56.3.9 string Qore::chr (any *arg*)

This function variant returns a string with a single ASCII NULL ('\0'); it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

Returns

a string with a single ASCII NULL ('\0')

43.56.3.10 nothing Qore::chr ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.11 string Qore::convert_encoding (string *str,* string *encoding*)

Performs explicit string character encoding conversions.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to convert
encoding	the encoding for the return value

Returns

a string corresponding to the string argument converted to the encoding given

Example:

```
my string $enc = convert_encoding($str, "iso-8859-1");
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

43.56.3.12 nothing Qore::convert_encoding ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.13 string Qore::f_printf (string *fmt,* ...)

Outputs the string passed to standard output, using the first argument as a format string; enforces field widths on arguments larger than the given field width.

This function will truncate output longer than any field width given in the format string

Restrictions:

Qore::PO_NO_TERMINAL_IO

Parameters

fmt	the format string
...	the argument(s) corresponding to format specifiers in the format string

Returns

the formatted string output corresponding to the arguments given

Example:

```
f_printf("%5s", "a long string"); # will print out "a lon"
```

See also

- String Formatting for information on the formatting string
- printf() for a similar function that does not enforce field widths

43.56.3.14 string Qore::f_printf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.15 string Qore::f_sprintf (string *fmt*, ...)

Returns a formatted string based on a format string and other arguments; enforces field widths on arguments larger than the given field width.

This function will truncate output longer than any field width given in the format string

Code Flags:

CONSTANT

Parameters

fmt	the format string
...	the argument(s) corresponding to format specifiers in the format string

Returns

a formatted string corresponding to the arguments given

Example:

```
my string $str = f_sprintf("%5s", "a long string"); # will return "a lon"
```

See also

- String Formatting for information on the formatting string
- sprintf() for a similar function that does not enforce field widths

43.56.3.16 string Qore::f_sprintf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.17 string Qore::f_vprintf (string *fmt*, any *varg*)

Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; enforces field widths on arguments larger than the given field width.

This function will truncate output longer than any field width given in the format string

Restrictions:

Qore::PO_NO_TERMINAL_IO

Parameters

fmt	the format string
varg	the argument(s) corresponding to format specifiers in the format string

Returns

the formatted string output corresponding to the arguments given

Example:

```
f_vprintf("%5s %3d", ("a long string", 5000)); # will print out "a lon 500"
```

See also

- String Formatting for information on the formatting string
- vprintf() for a similar function that does not enforce field widths

Since

Qore 0.8.4

43.56.3.18 string Qore::f_vsprintf (string *fmt*, any *varg*)

Returns a formatted string based on a format string and other arguments given as a list after the format string; enforces field widths on arguments larger than the given field width.

This function will truncate output longer than any field width given in the format string

Code Flags:

CONSTANT

Parameters

fmt	the format string
varg	the argument(s) corresponding to format specifiers in the format string

Returns

a formatted string corresponding to the arguments given

Example:

```
my string $str = f_vsprintf("%5s %3d\n", ("a lon", 500)); # will return "a lon 500"
```

See also

- String Formatting for information on the formatting string
- vsprintf() for a similar function that does not enforce field widths

Since

Qore 0.8.4

43.56.3.19 nothing Qore::flush ( )

Flushes output to the console output with print(), printf(), etc.

Restrictions:

Qore::PO_NO_TERMINAL_IO

Example:

```
flush();
```

43.56.3.20 string Qore::force_encoding (string *str,* string *encoding*)

Returns the first string argument tagged with the character encoding given as the second argument; does not actually change the string data.

Use only in the case that a string is tagged with the wrong encoding, for example, if a string from a File object has a different encoding than the File object.

Code Flags:

CONSTANT

Parameters

str	the string to be returned
encoding	the encoding to tag the return value with

Returns

the first string argument tagged with the character encoding given as the second argument; does not actually change the string data

Example:

```
my string $nstr = force_encoding($str, "iso-8859-1");
```

43.56.3.21 nothing Qore::force_encoding ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.22 string Qore::format_number (string *fmt*, softfloat *num*)

Returns a string of a formatted number according to a number argument and a format string.

Code Flags:

CONSTANT

Parameters

fmt	the format string has the following format: *<thousands_separator>* [*<decimal_separator><decimals>*] where: • *thousands_separator* and *decimal_separator* are single ASCII characters defining the thousands and decimal separator characters respectively, and • *decimals* is a single digit defining how may decimals should appear after the decimal point
num	the number to format

Returns

a string of a formatted number according to a number argument and a format string; if the format string does not follow the given format, then an empty string is returned

Example:

```
my string $nstr = format_number(".,3", -48392093894.2349); # returns "-48.392.093.894,235"
```

See also

Qore::zzz8floatzzz9::format()

43.56.3.23 nothing Qore::format_number ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.24 string Qore::get_encoding (string *str*)

Returns a string describing the character encoding of the string argument passed.

Code Flags:

CONSTANT

Parameters

str	the string to get the encoding for

Returns

a string describing the character encoding of the string argument passed

Example:

```
my string $enc = get_encoding($str);
```

Note

equivalent to Qore::zzz8stringzzz9::encoding()

43.56.3.25 nothing Qore::get_encoding ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.26 int Qore::index (softstring *str*, softstring *substr*, softint *pos* = 0)

Retrieves the character position of a substring within a string.

The *pos* argument and the return value are in character positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to search in
substr	the substring to find in *str*; if the character encoding of this string does not match *str*, then it will be converted to *str*'s character encoding before processing
pos	the starting character position for the search

Returns

the character position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = index($str, $substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::zzz8stringzzz9::find(softstring, softint)

See also

- Qore::zzz8stringzzz9::rfind(softstring, softint)
- rindex(softstring, softstring, softint)
- bindex(softstring, softstring, softint)
- brindex(softstring, softstring, softint)

43.56.3.27 int Qore::index ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.28 string Qore::join (string *str*, ...)

Creates a string from separator string and a list of arguments.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the separator string
...	the list of arguments to join; each argument will be converted to a string if necessary to be concatenated to the return value string; additionally if any string argument has a different character encoding than *str*, then it will be converted to *str*'s character encoding before concatentation to the return value string

Returns

a string created from a list and a seperator string, each element in the list will be present in the return value separated by the separator string

Example:

```
my string $str = join(":", "a", "b", "c"); # returns "a:b:c"
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

43.56.3.29 string Qore::join (string *str,* list *l*)

Creates a string from separator string and a list of arguments.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the separator string
l	the list of arguments to join; each argument will be converted to a string if necessary to be concatenated to the return value string; additionally if any string argument has a different character encoding than *str*, then it will be converted to *str*'s character encoding before concatentation to the return value string

Returns

a string created from a list and a separator string, each element in the list will be present in the return value separated by the separator string; the string returned with have the same character encoding as *str*

Example:

```
my string $str = join(":", ("a", "b", "c")); # returns "a:b:c"
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

Note

equivalent to Qore::zzz8listzzz9::join(string)

43.56.3.30 nothing Qore::join ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.31 int Qore::length (softstring *str*)

Returns the length in characters for the string passed.

Note that the byte length may differ from the character length with multi-byte character encodings. To get the byte length of a string, see strlen().

Code Flags:

CONSTANT

Parameters

str	the string to return the character length of

Returns

the length in characters for the string passed

Example:

```
my int $len = length($str);
```

Note

- this operation has *O(n)* complexity if called on a string with a multi-byte character encoding, otherwise it is *O(1)*
- equivalent to Qore::zzz8stringzzz9::length()

See also

- strlen()
- Qore::zzz8stringzzz9::strlen()
- Qore::zzz8stringzzz9::size()

43.56.3.32 int Qore::length (binary *bin*)

Returns the number of bytes in the binary object passed as an argument.

This variant of the function accepting a binary object as an argument works identically to the elements operator

Code Flags:

CONSTANT

Parameters

bin	the binary object to return the size of

Returns

the number of bytes in the binary object

Example:

```
my int $nbytes = length($bin);
```

43.56.3.33 nothing Qore::length ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.34 int Qore::length (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.35 int Qore::ord (softstring *str,* softint *offset* = 0)

Gives the numeric value of the given byte in the string passed; if no string is passed or the offset is after the end of the string, -1 is returned.

Code Flags:

CONSTANT

Parameters

str	the string containing the byte to be retrieved
offset	the byte offset of the byte to be retrieved

Returns

the numeric value of the given byte in the string passed; if the offset is after the end of the string or negative, -1 is returned

Example:

```
my int $i = ord($str);
```

Note

ord() only works on byte offsets and returns byte values

43.56.3.36 int Qore::ord ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.37 bool Qore::parse_boolean (string *str*)

tries to parse a string value as a boolean

Code Flags:

CONSTANT

Example:

```
my bool $b = parse_boolean($str);
```

Parameters

str	case-insensitive "on", "true", "enable*", "yes" are True, the rest is interpreted as a number where 0 is False, everything else is True

Returns

a boolean value according to the parsed argument

43.56.3.38 bool Qore::parse_boolean (...)

returns the first value passed as a boolean

Code Flags:

CONSTANT

Example:

```
my bool $b = parse_boolean($i);
```

Returns

the first value passed as a boolean

43.56.3.39 nothing Qore::print (...)

Outputs a string to standard output with no formatting.

Restrictions:

Qore::PO_NO_TERMINAL_IO

Parameters

...	each of the arguments passed to this function will be output literally with no output formatting

Example:

```
print("hello\n");
```

See also

printf() for a function that allows for formatted output

43.56.3.40 string Qore::printf (string *fmt*, ...)

Outputs the string passed to standard output, using the first argument as a format string; does not enforce field widths on arguments larger than the given field width.

This function will not truncate output longer than any field width given in the format string

Restrictions:

Qore::PO_NO_TERMINAL_IO

Parameters

fmt	the format string
...	the argument(s) corresponding to format specifiers in the format string

Returns

the formatted string output corresponding to the arguments given

Example:

```
printf("%5s", "a long string"); # will print out "a long string"
```

See also

- String Formatting for information on the formatting string
- f_printf() for a similar function that enforces field widths

43.56.3.41 string Qore::printf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_TERMINAL_IO

Code Flags:

NOOP

43.56.3.42 bool Qore::regex (string *str*, string *regex*, int *options* = 0)

Returns True if the regular expression matches the string passed, otherwise returns False.

Strings are converted to UTF-8 for pattern-matching; if any invalid encodings are encountered, an ENCODING-C↩ ONVERSION-ERROR is raised

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to test
regex	the regular expression pattern
options	regular expression options; see Regular Expression Constants for possible values

Returns

True if the regular expression matches the string passed, otherwise returns False

Example:

```
my bool $b = regex("hello", "^hel"); # returns True
```

Exceptions

REGEX-COMPILATION-ERROR	There was an error compiling the regular expression
REGEX-OPTION-ERROR	the option argument contains invalid option bits
ENCODING-CONVERSION-ERROR	this exception could be thrown if an encoding error is encountered when converting the given strings to UTF-8

Note

equivalent to Qore::zzz8stringzzz9::regex(string, int)

See also

Regular Expressions for more information about regular expression support in Qore

43.56.3.43 nothing Qore::regex ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.44 __7_ list Qore::regex_extract (string *str*, string *regex*, int *options* = 0)

Returns a list of substrings in a string based on matching patterns defined by a regular expression.

Strings are converted to UTF-8 for pattern-matching; if any invalid encodings are encountered, an ENCODING-CONVERSION-ERROR is raised

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to process
regex	the regular expression to use for matching, elements should be given in parentheses
options	regular expression options; see Regular Expression Constants for possible values

Returns

a list of substrings in a string based on matching patterns defined by a regular expression or NOTHING if no match was made

Example:

```
my *list $rv = regex_extract("ns:element", "(\\w+):(\\w+)");
```

Exceptions

REGEX-COMPILATION-ERROR	There was an error compiling the regular expression
REGEX-OPTION-ERROR	the option argument contains invalid option bits
ENCODING-CONVERSION-ERROR	this exception could be thrown if an encoding error is encountered when converting the given strings to UTF-8

Note

equivalent Qore::zzz8stringzzz9::regexExtract(string, int)

See also

Regular Expressions for more information about regular expression support in Qore

Since

Qore 0.8.8 this function accepts the Qore::RE_Global option to extract all occurrences of the pattern(s) in a string

43.56.3.45 nothing Qore::regex_extract ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.46 string Qore::regex_subst (string *str*, string *regex*, string *subst*, int *options* = `0`)

Returns a string with patterns substituted according to the arguments passed.

Strings are converted to UTF-8 for pattern-matching; if any invalid encodings are encountered, an ENCODING-CONVERSION-ERROR is raised

Code Flags:

RET_VALUE_ONLY

Parameters

str	the source string for substitution
regex	the regular expression pattern string
subst	the pattern for substitution; use `$1`, `$2`, etc for patterns in parentheses in the *regex* argument
options	regular expression options; see Regular Expression Constants for possible values

Returns

a string with patterns substituted according to the arguments passed

Example:

```
regex_subst("hello there", "^(.*) there$", "you $1"); #returns "you hello"
```

Example of global replacement:

```
my $content = "123
123
123";

printf("%s\n", regex_subst($content, "123", "456",
      Qore::RE_Global));

resulting in:
456
456
456
```

Exceptions

REGEX-COMPILATION-ERROR	There was an error compiling the regular expression
REGEX-OPTION-ERROR	the option argument contains invalid option bits
ENCODING-CONVERSION-ERROR	this exception could be thrown if an encoding error is encountered when converting the given strings to UTF-8

See also

- replace()
- Regular Expressions for more information about regular expression support in Qore

43.56.3.47 nothing Qore::regex_subst ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.48 string Qore::replace (string *str*, string *source*, string *target*, int *start* = 0, int *end* = -1)

Replaces all occurrences of a substring in a string with another string.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to process
source	the substring to replace; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
target	the replacement value for *source*; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
start	the starting character position in the source for the replacement where the first character is at position 0 (may not be the same as the byte position for multibyte character encodings)
end	the ending character position in the source for the replacement where the first character is at position 0 (may not be the same as the byte position for multibyte character encodings; that negative numbers mean to use the entire string)

Returns

a string with all occurrences of *source* replaced with *target*

Example:

```
my string $str = replace("hello there", "there", "you"); # returns "hello you"
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

Note

no regular expressions are used in this function, only direct replacements

See also

regex_subst()

43.56.3.49 nothing Qore::replace ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.50 string Qore::reverse (softstring *str*)

Reverses a string and returns the new string.

Works properly on strings with multi-byte character encodings as well (such as UTF-8)

Code Flags:

CONSTANT

Parameters

str	the string to reverse

Returns

a string with all characters in reverse order

Example:

```
my string $ns = reverse($str);
```

See also

reverse(list)

43.56.3.51 int Qore::rindex (softstring *str*, softstring *substr*, softint *pos* = -1)

Retrieves the character position of a substring within a string, starting the search from the end of the string.

The *pos* argument and the return value are in character positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to search in
substr	the substring to find in *str*; if the character encoding of this string does not match *str*, then it will be converted to *str*'s character encoding before processing
pos	the starting character position for the search, -1 means start from the end of the string

Returns

the character position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = rindex($str, $substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::zzz8stringzzz9::rfind(softstring, softint)

See also

- Qore::zzz8stringzzz9::find(softstring, softint)
- index(softstring, softstring, softint)
- bindex(softstring, softstring, softint)
- brindex(softstring, softstring, softint)

43.56.3.52 int Qore::rindex ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.53 list Qore::split (string *sep*, string *str*, bool *with_separator* = `False`)

Splits a string into a list of components based on a separator string.

Code Flags:

RET_VALUE_ONLY

Parameters

sep	the separator string; if the separator string is not found in the string to split, then a list with only one element containing the entire string argument is returned'; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
str	the string to split
with_separator	include the separator string in every element

Returns

a list of each component of a string separated by a separator string, with the separator removed; the separator pattern will not be included in the elements of the list returned unless the *with_separator* argument is True

Example:

```
my list $list = split(":", "some:text:here"); # returns ("some", "text", "here")
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

Note

equivalent to Qore::zzz8stringzzz9::split(string, bool)

43.56.3.54 list Qore::split (string *sep*, string *str*, string *quote*, bool *trim_unquoted* = `False`)

Splits a string into a list of components based on a separator string and a quote character.

The quote character can appear as the first part of a field, in which case it is assumed to designate the entire field. If instances of the quote character are found in the field preceded by a backquote character (`"\"`), then these quote characters are included as part of the field's text and not treated as quote characters

Code Flags:

RET_VALUE_ONLY

Parameters

sep	the separator string; if the separator string is not found in the string to split, then a list with only one element containing the entire string argument is returned'; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
str	the string to split
quote	the quote character

trim_unquoted	remove leading and trailing whitespace from unquoted fields

Returns

a list of each component of a string separated by a separator string, with the separator and any enclosing quote characters removed

Example:

```
my list $list = split(",", "some,'text with spaces',here", "'"); # returns ("some", "text with spaces"
      , "here")
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
SPLIT-ERROR	field missing closing quote character; extra text following quoted field

Note

equivalent to Qore::zzz8stringzzz9::split(string, string)

Since

Qore 0.8.6 the `trim_unquoted` parameter was added

43.56.3.55 list Qore::split (binary *sep*, binary *data*)

Returns a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed.

Code Flags:

CONSTANT

Parameters

data	the binary object to separate
sep	the bytes that separate the fields

Returns

a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed

Example:

```
my list $l = split($sep, $bin);
```

Note

equivalent to Qore::zzz8binaryzzz9::split(binary)

43.56.3.56 list Qore::split ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.57 string Qore::sprintf (string *fmt*, ...)

Returns a formatted string based on a format string and other arguments; does not enforce field widths on arguments larger than the given field width.

This function will not truncate output longer than any field width given in the format string

Code Flags:

CONSTANT

Parameters

fmt	the format string
...	the argument(s) corresponding to format specifiers in the format string

Returns

a formatted string corresponding to the arguments given

Example:

```
my string $str = sprintf("%5s", "a long string"); # will return "a long string"
```

See also

- String Formatting for information on the formatting string
- f_sprintf() for a similar function that enforces field widths

43.56.3.58 string Qore::sprintf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.59 int Qore::strlen (softstring *str*)

Returns the length in bytes of the string argument.

Note that the byte length may differ from the character length with multi-byte character encodings. For the character length of a string, see length().

Code Flags:

CONSTANT

Parameters

str	the string to return the byte length of

Returns

the length in bytes for the string passed

Example:

```
my int $size = strlen($str);
```

Note

equivalent to Qore::zzz8stringzzz9::strlen() and Qore::zzz8stringzzz9::size()

See also

- length()
- Qore::zzz8stringzzz9::length()

43.56.3.60 nothing Qore::strlen ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.61 int Qore::strlen (any *arg*)

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.62 string Qore::strmul (softstring *str*, softint *smul*, __7_ softint *offset*)

Returns a new string with a repeated string element and optionally removing trailing characters.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string element to repeat in the return value string
smul	the number of times the element should repeat
offset	the number of characters that should be removed from the end of the string after processing

Example:

```
my string $ret = strmul("%s,", 3, 1); # returns "%s,%s,%s"
```

Exceptions

STRMUL-ERROR	*offset* is < 0 or *smul* < 1

43.56.3.63 string Qore::substr (softstring *str*, softint *start*)

Returns a portion of a string starting from an integer offset.

Arguments can be negative, giving offsets from the end of the string. All offsets are character positions, not byte positions.

Code Flags:

RET_VALUE_ONLY

Parameters

str	The string to process
start	The starting character for the substring where the first character is at offset 0; if the offset is negative, it designates the number of characters from the end of the string. If the offset is 0, then the entire string is returned.

Returns

the substring of the string starting from an integer character offset; the rest of the string is returned after this offset; an empty string is returned if the argument cannot be satisfied

Example:

```
# get the last 10 characters of a string
my string $substr = substr($str, -10);
```

Exceptions

INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::zzz8stringzzz9::substr(softint)

43.56.3.64 string Qore::substr (softstring *str*, softint *start*, softint *len*)

Returns a portion of a string starting from an integer offset, with a length parameter.

Arguments can be negative, giving offsets from the end of the string. All offsets are character positions, not byte positions.

Code Flags:

RET_VALUE_ONLY

Parameters

str	The string to process
start	The starting character for the substring where the first character is at offset 0; if the offset is negative, it designates the number of characters from the end of the string
len	The maximum number of characters to copy; if this value is negative, the rest of the string from *start* will be copied to the substring, except without - *len* characters from the end of the string

Returns

the substring of the string according to the arguments passed; an empty string is returned if the argument cannot be satisfied

Example:

```
# get a substring 10 characters into the string except omitting the last 2 characters of the string
my string $substr = substr($str, 10, -2);
```

Exceptions

INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::zzz8stringzzz9::substr(softint, softint)

43.56.3.65 binary Qore::substr (binary *b*, softint *start*)

Returns a portion of a binary object starting from an integer offset.

Arguments can be negative, giving offsets from the end of the data.

Code Flags:

CONSTANT

Parameters

b	The binary object to process
start	The starting byte for the portion of the binary object where the first byte is at offset 0; if the offset is negative, it designates the number of bytes from the end of the data

Returns

the portion of the binary data argument starting from an integer byte offset; the rest of the data is returned after this offset

Example:

```
# get the last 10 bytes
my binary $b1 = substr($b, -10);
```

Note

equivalent to Qore::zzz8binaryzzz9::substr(softint)

Since

Qore 0.8.8

43.56.3.66 binary Qore::substr (binary *b*, softint *start*, softint *len*)

Returns a portion of a binary object starting from an integer offset, with a length parameter.

Arguments can be negative, giving offsets from the end of the data.

Code Flags:

CONSTANT

Parameters

b	The binary object to process
start	The starting byte for the portion of the binary object where the first byte is at offset 0; if the offset is negative, it designates the number of bytes from the end of the data
len	The maximum number of bytes to copy; if this value is negative, the rest of the data from *start* will be copied to the return value, except without - *len* bytes from the end of the data

Returns

the portion of the binary data argument according to the arguments passed

Example:

```
# get a binary object 10 characters into the data except omitting the last 2 bytes
my binary $b1 = substr($b, 10, -2);
```

Note

equivalent to Qore::zzz8binaryzzz9::substr(softint, softint)

Since

Qore 0.8.8

43.56.3.67 nothing Qore::substr ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.68 string Qore::tolower (string *str*)

Returns a string in all lower-case characters based on the argument passed.

Code Flags:

CONSTANT

Example:

```
my string $lwr = tolower($str);
```

This function operates on a very wide range of non-ASCII characters using a Unicode lookup table for mapping Latin, Cyrillic, Greek, Armenian, Georgian, etc characters.

Parameters

str	a string to process

Returns

a string in all lower-case characters based on the argument passed

Note

- equivalent to Qore::zzz8stringzzz9::lwr(string)

See also

- Qore::zzz8stringzzz9::lwr()
- Qore::zzz8stringzzz9::upr()
- toupper()

Since

Qore 0.8.8 this function operates on a wide range of characters and is no longer limited to ASCII characters

43.56.3.69 nothing Qore::tolower ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.70 string Qore::toupper (string *str*)

Returns a string in all upper-case characters based on the argument passed.

Code Flags:

CONSTANT

Example:

```
my string $upr = toupper($str);
```

This function operates on a very wide range of non-ASCII characters using a Unicode lookup table for mapping Latin, Cyrillic, Greek, Armenian, Georgian, etc characters.

Parameters

str	a string to process

Returns

a string in all upper-case characters based on the argument passed

Note

- equivalent to Qore::zzz8stringzzz9::upr(string)

See also

- Qore::zzz8stringzzz9::lwr()
- Qore::zzz8stringzzz9::upr()
- tolower()

Since

Qore 0.8.8 this function operates on a wide range of characters and is no longer limited to ASCII characters

43.56.3.71 nothing Qore::toupper ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.72 string Qore::trim (string *str*, __7_ string *chars*)

Removes byte characters from the start and end of a string and returns the new string (also see the trim operator)

By default (if the second argument is omitted or passed as an empty string) the following whitespace byte characters are removed: `" "`, `"\n"`, `"\r"`, `"\t"`, `"\v"` (vertical tab, ASCII 11), and `"\0"` (null character). To trim other characters, pass a string as the second argument specifying the byte characters to be removed.

Code Flags:

CONSTANT

Parameters

str	the string to trim
chars	the characters to trim from the start and end of the string (default: `" "`, `"\n"`, `"\r"`, `"\t"`, `"\v"` (vertical tab, ASCII 11), and `"\0"` (null character))

Returns

the trimmed string

Example:

```
my string $tstr = trim($str);
```

Bug it is not possible to trim multi-byte characters from strings using this function; each byte is treated as a character. No encoding conversions are done even if the *chars* argument has a different character encoding than the *str* argument.

43.56.3.73 __7_ string Qore::trim (reference *str,* __7_ string *chars*)

Removes byte characters from the start and end of a reference to an lvalue containing a string and returns string after processing (also see the trim operator)

By default (if the second argument is omitted or passed as an empty string) the following whitespace byte characters are removed: `" "`, `"\n"`, `"\r"`, `"\t"`, `"\v"` (vertical tab, ASCII 11), and `"\0"` (null character). To trim other characters, pass a string as the second argument specifying the byte characters to be removed.

This variant accepts variable references, in which case it will modify the string in place and also return the modified string.

Parameters

str	a reference to an lvalue continaing the string value to process; if the lvalue does not have a string value NOTHING is returned
chars	the characters to trim from the start and end of the string (default: `" "`, `"\n"`, `"\r"`, `"\t"`, `"\v"` (vertical tab, ASCII 11), and `"\0"` (null character))

Returns

the trimmed string

Example:

```
trim(\$str);
```

Bug it is not possible to trim multi-byte characters from strings using this function; each byte is treated as a character. No encoding conversions are done even if the *chars* argument has a different character encoding than the *str* argument.

43.56.3.74 nothing Qore::trim ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.56.3.75 string Qore::trunc_str (softstring *str,* softint *len,* __7_ string *encoding*)

Returns a truncated string with no more than the given number of bytes and optionally converted to a specific character encoding.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to truncate
len	the maximum byte length of the string
encoding	if given, the string returned will be returned in the given encoding, otherwise the encoding of \ str is used for the return value

Returns

the string truncated to be no longer than the given byte length in the character encoding given (or in the original string's character encoding if no *encoding* argument was passed to the function); if the string has a multi-byte character encoding, this function will guarantee that the returned string ends on a valid character (in this case the byte length of the string could be less than the limit given)

Example:

```
my string $column = trunc_str($str, 80, $dsp.getOSEncoding());
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if an explicit character encoding was given and an error occurs during encoding conversion

43.56.3.76 string Qore::vprintf (string *fmt*, ...)

Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; does not enforce field widths on arguments larger than the given field width.

This function will not truncate output longer than any field width given in the format string

Restrictions:

Qore::PO_NO_TERMINAL_IO

Parameters

fmt	the format string
...	the argument(s) corresponding to format specifiers in the format string

Returns

the formatted string output corresponding to the arguments given

Example:

```
vprintf("%5s %3d", ("a long string", 5000)); # will print out "a long string 5000"
```

See also

- String Formatting for information on the formatting string
- f_vprintf() for a similar function that enforces field widths

43.56.3.77 string Qore::vprintf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.56.3.78 string Qore::vsprintf (string *fmt,* any *varg*)

Returns a formatted string based on a format string and other arguments given as a list after the format string; does not enforce field widths on arguments larger than the given field width.

This function will not truncate output longer than any field width given in the format string

Code Flags:

CONSTANT

Parameters

fmt	the format string
varg	the argument(s) corresponding to format specifiers in the format string

Returns

a formatted string corresponding to the arguments given

Example:

```
my string $str = vsprintf("%5s %3d\n", ("a long string", 5000)); # will return "a long string 5000"
```

See also

- String Formatting for information on the formatting string
- f_vsprintf() for a similar function that enforces field widths

43.56.3.79 string Qore::vsprintf ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.57 Regular Expression Constants

Variables

- const Qore::RE_Caseless = PCRE_CASELESS

  Ignores case when matching regular expressions, equivalent to `/i`
- const Qore::RE_DotAll = PCRE_DOTALL

  makes a dot (.) match a newline character, equivalent to /s
- const Qore::RE_Extended = PCRE_EXTENDED

  ignores whitespace characters and enables comments prefixed by #, equivalent to `/x`
- const Qore::RE_Global = QRE_GLOBAL

  replace all matches globally in the string or extract all occurrences of the pattern(s) in the string, equivalent to `/g`
- const Qore::RE_MultiLine = PCRE_MULTILINE

  makes start-of-line (^) or end-of-line ($) match after or before any newline in the subject string, equivalent to `/m`

43.57.1 Detailed Description

The constants in this group can be combined with binary or to give regular expression options for the regex(), regex_subst(), and regex_extract() functions.

43.58 Threading Functions

Functions

- nothing Qore::delete_all_thread_data ()

  Deletes all keys in the thread-local data hash.
- nothing Qore::delete_thread_data (...)

  Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.
- nothing Qore::delete_thread_data (list l)

  Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.
- hash Qore::getAllThreadCallStacks ()

  Returns a hash of call stacks keyed by each TID (thread ID)
- hash Qore::get_all_thread_data ()

  Returns the entire thread-local data hash.
- any Qore::get_thread_data (string key)

  Returns the value of the thread-local data attached to the key passed.
- nothing Qore::get_thread_data ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ TimeZone Qore::get_thread_tz ()

  Returns any TimeZone set for the current thread, NOTHING if none is set.
- int Qore::gettid ()

  Returns the Qore thread ID (TID) of the current thread.
- nothing Qore::mark_thread_resources ()

  Marks thread resources so that any thread resources left allocated after this call will be cleaned up when throw_thread_resource_exceptions_to_mark() is called.
- int Qore::num_threads ()

  Returns the current number of threads in the process (not including the special signal handling thread)
- __7_ hash Qore::remove_thread_data (...)

  Removes the data associated to one or more keys in the thread-local data hash and returns the data removed.
- hash Qore::remove_thread_data (list l)

  Removes the data associated to one or more keys in the thread-local data hash from a literal list passed as the first argument and returns the data removed.
- nothing Qore::save_thread_data (hash h)

  Saves the data passed in the thread-local hash; all keys are merged into the thread-local hash, overwriting any information that may have been there before.
- nothing Qore::save_thread_data (string key, any value)

  Saves the data passed against the key passed as an argument in thread-local storage.
- nothing Qore::save_thread_data ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::set_thread_init (code init)

  Sets a call reference or closure to run every time a new thread is started.
- nothing Qore::set_thread_tz (TimeZone zone)

  Sets the default time zone for the current thread.
- nothing Qore::set_thread_tz ()

  Clears the thread-local time zone for the current thread; after this call TimeZone::get() will return the value set for the current Program.
- list Qore::thread_list ()

  Returns a list of all current thread IDs.

- nothing Qore::throwThreadResourceExceptions ()

  Immediately runs all thread resource cleanup routines for the current thread and throws all associated exceptions.
- bool Qore::throw_thread_resource_exceptions_to_mark ()

  Immediately runs all thread resource cleanup routines for the current thread for thread resources created since the last call to mark_thread_resources() and throws all associated exceptions.

43.58.1 Detailed Description

Threading functions

43.58.2 Function Documentation

43.58.2.1 nothing Qore::delete_all_thread_data ( )

Deletes all keys in the thread-local data hash.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Example:

```
delete_all_thread_data();
```

Note

This function does not throw any exceptions, however if an object is deleted from the thread-local data hash, then it could throw an exception in its destructor

43.58.2.2 nothing Qore::delete_thread_data (...)

Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

...	Deletes the data associated to one or more keys in the thread-local data hash corresponding to each string argument in the top-level argument list; arguments are converted to strings if necessary

Example:

```
delete_thread_data("key1", "key2");
```

Note

This function does not throw any exceptions, however if an object is deleted from the thread-local data hash, then it could throw an exception in its destructor

See also

remove_thread_data() for a similar function that does not explicitly destroy objects in the thread-local data hash

43.58.2.3 nothing Qore::delete_thread_data (list *l*)

Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

l	a list of key names for deleting keys in the thread-local data hash; values are converted to strings if necessary

Example:

```
delete_thread_data($list_of_keys);
```

Note

This function does not throw any exceptions, however if an object is deleted from the thread-local data hash, then it could throw an exception in its destructor

See also

remove_thread_data() for a similar function that does not explicitly destroy objects in the thread-local data hash

43.58.2.4 hash Qore::get_all_thread_data ( )

Returns the entire thread-local data hash.

Returns

the entire thread-local data hash

Restrictions:

Qore::PO_NO_THREAD_CONTROL, Qore::PO_NO_THREAD_INFO

Code Flags:

CONSTANT

Example:

```
my hash $th = get_all_thread_data();
```

43.58.2.5 any Qore::get_thread_data (string *key*)

Returns the value of the thread-local data attached to the key passed.

Returns

the value of the thread-local data attached to the key passed

Restrictions:

Qore::PO_NO_THREAD_CONTROL, Qore::PO_NO_THREAD_INFO

Code Flags:

CONSTANT

Example:

```
my any $data = get_thread_data("key1");
```

43.58.2.6 nothing Qore::get_thread_data ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_THREAD_CONTROL, Qore::PO_NO_THREAD_INFO

Code Flags:

RUNTIME_NOOP

43.58.2.7 __7_ TimeZone Qore::get_thread_tz ( )

Returns any TimeZone set for the current thread, NOTHING if none is set.

Returns

any TimeZone set for the current thread, NOTHING if none is set

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Example:

```
my *TimeZone $tz = get_thread_tz();
set_thread_tz(new TimeZone("Europe/Prague"));
on_exit set_thread_tz($tz);
```

See also

- set_thread_tz(TimeZone)
- set_thread_tz()

Since

Qore 0.8.5

43.58.2.8 hash Qore::getAllThreadCallStacks ( )

Returns a hash of call stacks keyed by each TID (thread ID)

Platform Availability:

Qore::Option::HAVE_RUNTIME_THREAD_STACK_TRACE

Restrictions:

Qore::PO_NO_THREAD_CONTROL, Qore::PO_NO_THREAD_INFO

Returns

a hash of call stacks keyed by each TID (thread ID); see call stacks for the format of the hash values

Example:

```
my hash $cs = getAllThreadCallStacks();
foreach my string $tid in ($cs.keyIterator()) {
    printf("TID %d\n", $tid);
    my int $i;
    foreach my hash $l in ($cs.$tid)
        if ($l.type != "new-thread")
            printf(" %d: %s() called at %s:%d (%s function)\n", ++$i, $l.function, $l.file, $l.line,
  $l.type);
        else
            printf(" %d: *** thread started by background operator ***\n", ++$i);
}
```

43.58.2.9 int Qore::gettid ( )

Returns the Qore thread ID (TID) of the current thread.

Returns

the Qore thread ID (TID) of the current thread

Restrictions:

Qore::PO_NO_THREAD_INFO

Code Flags:

CONSTANT

Example:

```
my int $tid = gettid();
```

43.58.2.10 nothing Qore::mark_thread_resources ( )

Marks thread resources so that any thread resources left allocated after this call will be cleaned up when throw_↩ thread_resource_exceptions_to_mark() is called.

When exceptions are thrown by this function, thread-local resources are also cleaned up at the same time.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Example:

```
try {
    mark_thread_resources();

    # calling the following function will ensure that any thread-resources
    # allocated since the last call to mark_thread_resources() will be cleaned
    # up and associated exceptions will be thrown
    on_exit throw_thread_resource_exceptions_to_mark();

    # ... some code or calls that may allocate thread resources
}
catch ($ex) {
    # ... log or handle exceptions
}
```

See also

throwThreadResourceExceptions(), mark_thread_resources()

Since

Qore 0.8.4

43.58.2.11 int Qore::num_threads ( )

Returns the current number of threads in the process (not including the special signal handling thread)

Returns

the current number of threads in the process (not including the special signal handling thread)

Restrictions:

Qore::PO_NO_THREAD_INFO

Code Flags:

CONSTANT

Example:

```
my int $num = num_threads();
```

43.58.2.12 __7_ hash Qore::remove_thread_data (...)

Removes the data associated to one or more keys in the thread-local data hash and returns the data removed.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

...	the key names should be given directly in the argument list with this variant. If the given hash keys do not exist in the thread-local data hash, then the given key in the return value will have no value assigned

Returns

a hash of the data removed or NOTHING if no arguments were passed to the function

Example:

```
my hash $h = remove_thread_data("filename");
```

Since

Qore 0.8.4 this function returns the values it removes

43.58.2.13 hash Qore::remove_thread_data (list *l*)

Removes the data associated to one or more keys in the thread-local data hash from a literal list passed as the first argument and returns the data removed.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

l	a list of key names to remove from the thread-local data hash. If the given hash keys do not exist in the thread-local data hash, then the given key in the return value will have no value assigned

Returns

a hash of the data removed

Example:

```
my hash $h = remove_thread_data($key_list);
```

Since

Qore 0.8.4 this function returns the values it removes

43.58.2.14 nothing Qore::save_thread_data (hash *h*)

Saves the data passed in the thread-local hash; all keys are merged into the thread-local hash, overwriting any information that may have been there before.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

h	a hash of data to save in the thread-local data hash

Example:

```
save_thread_data($h);
```

Note

This function does not throw any exceptions, however if a value is removed from the thread-local data hash by being overwritten with a new value, and the value is an object that goes out of scope, then such an object could throw an exception in its destructor

43.58.2.15 nothing Qore::save_thread_data (string *key,* any *value*)

Saves the data passed against the key passed as an argument in thread-local storage.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Parameters

key	the name of the key in the thread-local hash to save the data against
value	the value to save in the thread-local hash against the key

Example:

```
save_thread_data("key1", $value);
```

Note

This function does not throw any exceptions, however if a value is removed from the thread-local data hash by being overwritten with a new value, and the value is an object that goes out of scope, then such an object could throw an exception in its destructor

43.58.2.16 nothing Qore::save_thread_data ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Code Flags:

RUNTIME_NOOP

43.58.2.17 bool Qore::set_thread_init (code *init*)

Sets a call reference or closure to run every time a new thread is started.

This code can be used to initialize global thread-local variables, for example.

Returns

True if there was already user initialization code set, False if not

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Example:

```
set_thread_init(sub () { $var = 123; });
```

Note

the code will be run for all new threads, but is not run by this function for the current thread

43.58.2.18 nothing Qore::set_thread_tz (TimeZone *zone*)

Sets the default time zone for the current thread.

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Parameters

zone	the TimeZone object for the current thread

This will cause the TimeZone::get() method to return the given TimeZone when called from the same thread

Example:

```
my *TimeZone $tz = get_thread_tz();
set_thread_tz(new TimeZone("Europe/Prague"));
on_exit set_thread_tz($tz);
```

Note

The TimeZone will only be set for the current thread in the current Program

See also

- set_thread_tz()
- get_thread_tz()

Since

Qore 0.8.5

43.58.2.19 nothing Qore::set_thread_tz ( )

Clears the thread-local time zone for the current thread; after this call TimeZone::get() will return the value set for the current Program.

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Example:

```
my *TimeZone $tz = get_thread_tz();
set_thread_tz(new TimeZone("Europe/Prague"));
on_exit set_thread_tz($tz);
```

Note

The TimeZone will only be cleared in the current thread in the current Program

See also

- set_thread_tz(TimeZone)
- get_thread_tz()

Since

Qore 0.8.5

43.58.2.20 list Qore::thread_list ( )

Returns a list of all current thread IDs.

Note that the special signal handling thread with TID 0 is never included in the list returned by this function

Returns

a list of all current thread IDs

Restrictions:

Qore::PO_NO_THREAD_INFO

Code Flags:

CONSTANT

Example:

```
my list $l = thread_list();
```

43.58.2.21 bool Qore::throw_thread_resource_exceptions_to_mark ( )

Immediately runs all thread resource cleanup routines for the current thread for thread resources created since the last call to mark_thread_resources() and throws all associated exceptions.

When exceptions are thrown by this function, thread-local resources are also cleaned up at the same time.

Returns

True if there are additional thread resource marks to process, False if there are no more

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Example:

```
try {
    mark_thread_resources();
    # ... some code or calls that may allocate thread resources

    # calling the following will ensure that any thread-resources allocated since
    # the last call to mark_thread_resources() will be cleaned up and associated
    # exceptions will be thrown
    throw_thread_resource_exceptions_to_mark();
}
catch ($ex) {
    # ... log or handle exceptions
}
```

See also

throwThreadResourceExceptions(), mark_thread_resources()

Since

Qore 0.8.4

43.58.2.22 nothing Qore::throwThreadResourceExceptions ( )

Immediately runs all thread resource cleanup routines for the current thread and throws all associated exceptions.

This function is particularly useful when used in combination with embedded code in order to catch (and log, for example) thread resource errors (ex: uncommitted transactions, unlocked locks, etc) - this can be used when control returns to the "master" program to ensure that no thread-local resources have been left active.

This function will run all thread resource cleanup routines even if mark_thread_resources() has been called (i.e. it clears all marks as well).

When exceptions are thrown by this function, thread-local resources are also cleaned up at the same time.

Restrictions:

Qore::PO_NO_THREAD_CONTROL

Example:

```
try {
    throwThreadResourceExceptions();
}
catch ($ex) {
    # ... log or handle exceptions
}
```

See also

mark_thread_resources(), throw_thread_resource_exceptions_to_mark()

43.59 Date and Time Functions

Functions

- int Qore::clock_getmicros ()

  Returns an integer representing the system time in microseconds (1/1000000 second intervals) since Jan 1, 1970 00:00:00Z.
- int Qore::clock_getmillis ()

  Returns an integer representing the system time in milliseconds (1/1000 second intervals since Jan 1, 1970 00:00)
- int Qore::clock_getnanos ()

  Returns an integer representing the system time in nanoseconds (1/1000000000 second intervals) since Jan 1, 1970 00:00:00Z.
- date Qore::date (date dt)

  Returns the date passed.
- date Qore::date (string dtstr)

  Converts the argument to a date and returns the date.
- date Qore::date (float f)

  The argument is assumed to be the number of seconds and fractions of a second since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.
- date Qore::date (softint i)

  The argument is assumed to be the number of seconds since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.
- date Qore::date ()

  This function just returns 1970-01-01Z.
- date Qore::date (null null)

  This function just returns 1970-01-01Z.
- date Qore::date (string dtstr, string mask)

  Returns the date/time value corresponding to parsing a string argument according to a format mask.
- hash Qore::date_info (date dt)

  Returns a hash of broken-down date/time information for the given date argument (can be either a relative or absolute date)
- hash Qore::date_info ()

  Returns a hash of broken-down date/time information for the current date and time.
- date Qore::date_ms (softint ms)

  Converts an integer argument representing the offset in milliseconds from January 1, 1970 in the local time zone to a date in the local time zone.
- nothing Qore::date_ms ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::date_us (softint us)

  Converts an integer argument representing the offset in microseconds from January 1, 1970 in the local time zone to a date in the local time zone.
- date Qore::days (softint days)

  Returns a relative date/time value in days based on the integer argument passed to be used in date arithmetic.
- nothing Qore::days ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string Qore::format_date (string format, date dt)

  Returns a formatted string for a date argument passed.
- nothing Qore::format_date ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::getDateFromISOWeek (softint year, softint week, softint day=1)

Retuns an absolute date value for the ISO-8601 calendar week information passed (year, week number, optional: weekday, where 1=Monday, 7=Sunday) in the current time zone.

- int Qore::getDayNumber (date dt)

  Returns an integer representing the ordinal day number in the year (corresponding to the ISO-8601 day number) for the absolute date value passed.

- nothing Qore::getDayNumber ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::getDayOfWeek (date dt)

  Returns an integer representing the day of the week for the absolute date value passed (0=Sunday, 6=Saturday)

- nothing Qore::getDayOfWeek ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::getISODayOfWeek (date dt)

  Returns an integer representing the ISO-8601 day of the week for the absolute date value passed (1=Monday, 7=Sunday)

- nothing Qore::getISODayOfWeek ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- hash Qore::getISOWeekHash (date dt)

  Returns a hash representing the ISO-8601 calendar week information for the absolute date passed (hash keys: "year", "week", "day")

- nothing Qore::getISOWeekHash ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string Qore::getISOWeekString (date dt)

  Returns a string representing the ISO-8601 calendar week information for the absolute date passed (ex: 2006-01-01 = "2005-W52-7")

- nothing Qore::getISOWeekString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::get_days (date dt)

  Returns an integer corresponding to the literal day value in the date (does not calculate a duration)

- nothing Qore::get_days ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::get_duration_microseconds (date dt)

  Returns an integer value representing the the number of microseconds of duration in the value of the date passed (can be either a relative or absolute date)

- int Qore::get_duration_milliseconds (date dt)

  Returns an integer value representing the the number of milliseconds of duration in the value of the date passed (can be either a relative or absolute date)

- int Qore::get_duration_seconds (date dt)

  Returns an integer value representing the the number of seconds of duration in the value of the date passed (can be either a relative or absolute date)

- int Qore::get_epoch_seconds (date dt)

  Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.

- nothing Qore::get_epoch_seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::get_hours (date dt)

  Returns an integer corresponding to the literal hour value in the date (does not calculate a duration)

- nothing Qore::get_hours ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int Qore::get_microseconds (date dt)

  Returns an integer corresponding to the literal microsecond value in the date (does not calculate a duration)
- date Qore::get_midnight (date dt)

  Returns midnight on the date passed (strips the time component on the new value)
- nothing Qore::get_midnight ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::get_milliseconds (date dt)

  Returns an integer corresponding to the literal millisecond value in the date (does not calculate a duration)
- nothing Qore::get_milliseconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::get_minutes (date dt)

  Returns an integer corresponding to the literal minute value in the date (does not calculate a duration)
- nothing Qore::get_minutes ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::get_months (date dt)

  Returns an integer corresponding to the literal month value in the date (does not calculate a duration)
- nothing Qore::get_months ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::get_seconds (date dt)

  Returns an integer corresponding to the literal second value in the date (does not calculate a duration)
- nothing Qore::get_seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::get_years (date dt)

  Returns an integer corresponding to the literal year value in the date (does not calculate a duration)
- nothing Qore::get_years ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::gmtime ()

  Returns the current UTC (GMT) time with a resolution of a second.
- date Qore::gmtime (softint secs, softint us=0)

  Returns a date/time value in UTC (GMT) from arguments giving the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)
- date Qore::gmtime (date dt)

  Returns the date and time in UTC (GMT) corresponding to the date argument passed.
- date Qore::hours (softint hours)

  Returns a relative date/time value in hours based on the integer argument passed to be used in date arithmetic.
- nothing Qore::hours ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::is_date_absolute (date dt)

  Returns True if the argument is an absolute date/time value, False if not.
- bool Qore::is_date_absolute ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool Qore::is_date_relative (date dt)

  Returns True if the argument is an relative date/time value, False if not.

- bool Qore::is_date_relative ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::localtime ()

  Returns the current date and time with a resolution to the second.
- date Qore::localtime (softint secs, softint us=0)

  Returns the date and time in the local time zone corresponding to the integer arguments passed, which are interpreted as the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)
- date Qore::localtime (date dt)

  Returns the date and time in the local time zone corresponding to the date argument passed.
- date Qore::microseconds (softint us)

  Returns a relative date/time value in microseconds based on the integer argument passed to be used in date arithmetic.
- date Qore::milliseconds (softint ms)

  Returns a relative date/time value in milliseconds based on the integer argument passed to be used in date arithmetic.
- nothing Qore::milliseconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::minutes (softint minutes)

  Returns a relative date/time value in minutes based on the integer argument passed to be used in date arithmetic.
- nothing Qore::minutes ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::mktime (date dt)

  Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.
- nothing Qore::mktime ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::months (softint months)

  Returns a relative date/time value in months based on the integer argument passed to be used in date arithmetic.
- nothing Qore::months ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::now ()

  Returns the current date and time with a resolution to the second.
- date Qore::now_ms ()

  Returns the current date and time with a resolution to the millisecond.
- date Qore::now_us ()

  Returns the current date and time with a resolution to the microsecond.
- date Qore::now_utc ()

  Returns the current UTC date and time with a resolution to the microsecond.
- date Qore::seconds (softint seconds)

  Returns a relative date/time value in seconds based on the integer argument passed to be used in date arithmetic.
- nothing Qore::seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int Qore::timegm (date dt)

  Returns the number of seconds since January 1, 1970 00:00:00 in the local time zone for the given date.
- nothing Qore::timegm ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date Qore::years (softint years)

Returns a relative date/time value in years based on the integer argument passed to be used in date arithmetic.

- nothing Qore::years ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

43.59.1 Detailed Description

43.59.2 Date Formatting Codes

Format Code	Description
YY	last two digits of year
YYYY	four-digit year
M	non zero-padded month number (1-12)
MM	zero-padded two-digit month number (01-12)
Month	long month string (ex: "January")
MONTH	long month string capitalized (ex: "JANUARY")
Mon	abbreviated month (ex: "Jan")
MON	abbreviated month, capitalized (ex: "JAN")
D	non zero-padded day number (1 - 31)
DD	zero-padded day number (01 - 31)
Day	long day of week string (ex: "Monday")
DAY	long day of week string, capitalized (ex: "MONDAY")
Dy	abbreviated day of week string (ex: "Mon")
DY	abbreviated day of week string capitalized (ex: "MON")
H	non zero-padded hour number (0 - 23)
HH	zero-padded hour number (00 - 23)
h	non zero-padded hour number, 12-hour clock (1 - 12)
hh	zero-padded hour number, 12-hour clock (01 - 12)
m	non zero-padded minute number (0 - 59)
mm	zero-padded minute number (00 - 59)
S	non zero-padded second number (0 - 59)
SS	zero-padded second number (00 - 59)
P	"AM" or "PM" (upper-case)
p	"am" or "pm" (lower-case)
u	non zero-padded millisecond number (0 - 999)
uu or ms	zero-padded millisecond number (000 - 999)
x	non zero-padded microsecond number (0 - 999999)
xx or us	zero-padded microsecond number (000000 - 999999)
y	microseconds, with trailing zeros removed (suitable for use after the '.')
z	local time zone name (ex: "EST") if available, otherwise the UTC offset (ex: "+01:00")
Z	time zone UTC offset like +HH:mm[:SS] (ex: "+01:00"), seconds are only included if non-zero

Since

"us" was added in Qore 0.8.7

Bug the "u" and "uu" format codes work with milliseconds and not microseconds since they were implemented before qore supported microsecond time resolution; this cannot be changed without breaking backwards-compatibility

43.59.3 Date/Time Information Hash

Key	Type	Absolute/Relative	Description
relative	bool	Both	True if the date is a relative date, False if it is absolute
year	int	Both	The year value of the date
month	int	Both	The month value of the date
day	int	Both	The day value of the date (day of the month for absolute dates)
hour	int	Both	The hour value of the date
minute	int	Both	The minute value of the date
second	int	Both	The second value of the date
microsecond	int	Both	The microsecond value of the date
dow	int	Absolute Only	The day of the week, where 0=Sunday, 1=Monday, ... 6=Saturday
doy	int	Absolute Only	The ordinal day number in the year
utc_secs_east	int	Absolute Only	Offset from UTC in seconds east; if the time zone is west of UTC then the value will be negative
dst	bool	Absolute Only	A flag if daylight savings time is in effect
zone	TimeZone	Absolute Only	The time zone for the time
zone_name	string	Absolute Only	The name of the time zone for the given time (ex: "CEST" for Central European Summer Time for a time during summer time or "CET" for Central European Time for the same time zone while daylight savings time is not active)

43.59.4 Date Mask Format

The date mask (as used in Qore::date(string, string) and Qore::TimeZone::date(string, string)) is used to specify the string format of a date/time value for parsing to a date/time value.

The fields in the mask string are as follows:

- `YY`: The last 2 digits of the year; the current century is assumed
- `YYYY`: A 4-digit year value (ex: 1956, 0870)
- `MM`: A 2-digit month number (ex: 01, 11)
- `Mon` or `MON`: A 3-character English month name abbreviation (parsing is case-insensitive: ex: `"Apr"` or `"APR"` or `"apr"` for April)

- `DD`: A 2-digit day number (ex: 29, 04)
- `HH`: A 2-digit hour number (ex: 23, 04)
- `mm`: A 2-digit minute number (ex: 11, 04)
- `ms`: A 3-digit millisecond number (ex: 511, 042)
- `SS`: A 2-digit second number (ex: 11, 04)
- `sss`: A 3-digit millisecond number (ex: 511, 042)
- `ssssss`: A 6-digit microsecond number (ex: 240511, 005142)
- `us`: A 6-digit microsecond number (ex: 240511, 005142)

Bug currently only English month abbreviations are accepted in the date/time mask

43.59.5 Function Documentation

43.59.5.1 int Qore::clock_getmicros ( )

Returns an integer representing the system time in microseconds (1/1000000 second intervals) since Jan 1, 1970 00:00:00Z.

Returns

an integer representing the system time in microseconds (1/1000000 second intervals) since Jan 1, 1970 00:00:00Z

Code Flags:

CONSTANT

Example:

```
my int $us = clock_getmicros();
```

See also

- clock_getmillis()
- clock_getmicros()

43.59.5.2 int Qore::clock_getmillis ( )

Returns an integer representing the system time in milliseconds (1/1000 second intervals since Jan 1, 1970 00:00)

Returns

an integer representing the system time in milliseconds (1/1000 second intervals since Jan 1, 1970 00:00)

Code Flags:

CONSTANT

Example:

```
my int $ms = clock_getmillis();
```

See also

- clock_getnanos()
- clock_getmicros()

43.59.5.3 int Qore::clock_getnanos ( )

Returns an integer representing the system time in nanoseconds (1/1000000000 second intervals) since Jan 1, 1970 00:00:00Z.

Returns

an integer representing the system time in nanoseconds (1/1000000000 second intervals) since Jan 1, 1970 00:00:00Z

Code Flags:

CONSTANT

Example:

```
my int $ns = clock_getnanos();
```

See also

- clock_getmillis()
- clock_getmicros()

43.59.5.4 date Qore::date (date *dt*)

Returns the date passed.

This function is included because the date() function is used to convert values to dates (similar to a C/C++ cast)

Code Flags:

CONSTANT

Parameters

dt	the date to return

Returns

the date passed

43.59.5.5 date Qore::date (string *dtstr*)

Converts the argument to a date and returns the date.

Code Flags:

CONSTANT

Parameters

dtstr	string a string to bo converted literally to an absolute date/time value; if no time zone information is present, the local time zone is assumed

Returns

the absolute date/time value corresponding to the string argument passed

Example:

```
my date $d = date("2001-01-01T15:35:23"); # returns 2001-01-01 15:35:23 Mon +01:00 (CET)
my date $d = date("20010101 15:35:23Z"); # returns 2001-01-01 15:35:23 Mon Z (UTC)
my date $d = date("20010101 153523Z"); # returns 2001-01-01 15:35:23 Mon Z (UTC)
my date $d = date("20010101 153523-02"); # returns 2001-01-01 15:35:23 Mon -02:00 (-02:00)
my date $d = date("20010101 153523-02:00:00"); # returns 2001-01-01 15:35:23 Mon -02:00 (-02:00)
my date $d = date("2001-01-01-153523-020000"); # returns 2001-01-01 15:35:23 Mon -02:00 (-02:00)
my date $d = date("20010101-153523"); # returns 2001-01-01 15:35:23 Mon +01:00 (CET)
```

See also

date(string, string)

43.59.5.6 date Qore::date (float *f*)

The argument is assumed to be the number of seconds and fractions of a second since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.

Code Flags:

CONSTANT

Parameters

f	the number of seconds and fractions of a second since 1970-01-01 in the local time zone

Returns

a date corresponding to the number of seconds and fractions of a second since 1970-01-01 in the local time zone passed as the sole argument

Example:

```
my date $d = date($f);
```

43.59.5.7 date Qore::date (softint *i*)

The argument is assumed to be the number of seconds since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.

Code Flags:

CONSTANT

Parameters

i	the number of seconds since 1970-01-01 in the local time zone

Returns

a date corresponding to the number of seconds since 1970-01-01 in the local time zone passed as the sole argument

Example:

```
my date $d = date($i);
```

43.59.5.8 date Qore::date ( )

This function just returns 1970-01-01Z.

Code Flags:

CONSTANT

This function is included because the date() function is used to convert values to dates (similar to a C/C++ cast)

Returns

1970-01-01Z

43.59.5.9 date Qore::date (null *null*)

This function just returns 1970-01-01Z.

Code Flags:

CONSTANT

This function is included because the date() function is used to convert values to dates (similar to a C/C++ cast)

Returns

1970-01-01Z

43.59.5.10 date Qore::date (string *dtstr*, string *mask*)

Returns the date/time value corresponding to parsing a string argument according to a format mask.

Code Flags:

RET_VALUE_ONLY

Parameters

dtstr	a string giving a date
mask	the mask for the date value; see Date Mask Format for information on the format of the format mask

Returns

the date/time value corresponding to parsing the *dtstr* string argument according to *mask* serving as a format mask

Example:

```
my date $dt = date("20100401 234520 230394", "YYYYMMDD HHmmSS ssssss"); # returns 2010-04-01T23:45:
      20.230394
```

Exceptions

DATE-CONVERT-ERROR	invalid mask specification

See also

Qore::TimeZone::date(string, string) for a method that allows the date to be parsed in a particular time zone

43.59.5.11 hash Qore::date_info (date *dt*)

Returns a hash of broken-down date/time information for the given date argument (can be either a relative or absolute date)

Code Flags:

CONSTANT

Parameters

dt	the date to return information for

Returns

a hash of broken-down date/time information for the given date argument

Example:

```
my hash $h = date_info($date);
```

Note

equivalent to Qore::zzz8datezzz9::info()

43.59.5.12 hash Qore::date_info ( )

Returns a hash of broken-down date/time information for the current date and time.

Returns

a hash of broken-down date/time information for the current date and time

Code Flags:

CONSTANT

Example:

```
my hash $h = date_info();
```

43.59.5.13 date Qore::date_ms (softint *ms*)

Converts an integer argument representing the offset in milliseconds from January 1, 1970 in the local time zone to a date in the local time zone.

Code Flags:

CONSTANT

Parameters

ms	an integer argument representing the offset in milliseconds from January 1, 1970 in the local time zone

Returns

a date in the local time zone corresponding to the argument passed which is an offset in milliseconds from January 1, 1970 in the local time zone

Example:

```
my date $date = date_ms(1); # returns 1970-01-01T00:00:00.001 in the local time zone
```

See also

date_us(softint)

43.59.5.14 nothing Qore::date_ms ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.15 date Qore::date_us (softint *us*)

Converts an integer argument representing the offset in microseconds from January 1, 1970 in the local time zone to a date in the local time zone.

Code Flags:

CONSTANT

Parameters

us	an integer argument representing the offset in microseconds from January 1, 1970 in the local time zone

Returns

a date in the local time zone corresponding to the argument passed which is an offset in microseconds from January 1, 1970 in the local time zone

Example:

```
my date $date = date_us(1); # returns 1970-01-01T00:00:00.000001 in the local time zone
```

See also

date_ms(softint)

43.59.5.16 date Qore::days (softint *days*)

Returns a relative date/time value in days based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

days	the number of days to return

Returns

a relative date/time value in days based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = days(100);
```

See also

- years(softint)
- months(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.17 nothing Qore::days ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.18 string Qore::format_date (string *format,* date *dt*)

Returns a formatted string for a date argument passed.

Code Flags:

CONSTANT

Parameters

format	a string giving the format for the date; see Date Formatting Codes for more information about this string
dt	the date to use for the string output

Returns

a formatted string for a date argument passed

Example:

```
my string $str = format_date("Day, Mon D, YYYY-MM-DD HH:mm:SS", 2004-02-01T12:30:00);
# returns "Sunday, Feb 1, 2004-02-01 12:30:00"
```

Bug there is no locale support; day and month names and abbreviations are only returned in English

43.59.5.19 nothing Qore::format_date ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.20 int Qore::get_days (date *dt*)

Returns an integer corresponding to the literal day value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the day value in the date

Example:

```
my int $n = get_days($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::days()

43.59.5.21 nothing Qore::get_days ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.22 int Qore::get_duration_microseconds (date *dt*)

Returns an integer value representing the the number of microseconds of duration in the value of the date passed (can be either a relative or absolute date)

Code Flags:

CONSTANT

Parameters

dt	a relative or absolute date argument

Returns

an integer value representing the the number of microseconds of duration in the value of the date passed; if the argument is a relative date, the value is normalized to microseconds and the integer microseconds value is returned, if the argument is an absolute date, the duration in microseconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of microseconds from the present time to the future date is returned, if an absolute date in the past is passed as an argument, also 0 is returned (the function does not calculate microsecond differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Example:

```
my int $us = get_duration_microseconds(PT2M15S3u); # returns 135000003
```

Note

- equivalent to Qore::zzz8datezzz9::durationMicroseconds()
- to get the literal microseconds integer value from a date/time value without calculating a duration, use get_microseconds(date)

See also

- get_duration_seconds()
- get_duration_milliseconds()

43.59.5.23 int Qore::get_duration_milliseconds (date *dt*)

Returns an integer value representing the the number of milliseconds of duration in the value of the date passed (can be either a relative or absolute date)

The duration in milliseconds is calculated and any fractional milliseconds are truncated (no rounding is performed)

Code Flags:

CONSTANT

Parameters

dt	a relative or absolute date argument

Returns

an integer value representing the the number of milliseconds of duration in the value of the date passed; if the argument is a relative date, the value is normalized to milliseconds and the integer milliseconds value is returned, if the argument is an absolute date, the duration in milliseconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of milliseconds from the present time to the future date is returned, if an absolute date in the past is passed as an argument, also 0 is returned (the function does not calculate millisecond differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Example:

```
my int $ms = get_duration_milliseconds(PT2M15S3u); # returns 135000
```

Note

- equivalent to Qore::zzz8datezzz9::durationMilliseconds()
- to get the literal milliseconds integer value from a date/time value without calculating a duration, use get_milliseconds(date)

See also

- get_duration_seconds()
- get_duration_microseconds()

43.59.5.24 int Qore::get_duration_seconds (date *dt*)

Returns an integer value representing the the number of seconds of duration in the value of the date passed (can be either a relative or absolute date)

The duration in seconds is calculated and any fractional seconds are truncated (no rounding is performed)

Code Flags:

CONSTANT

Parameters

dt	a relative or absolute date argument

Returns

an integer value representing the the number of seconds of duration in the value of the date passed; if the argument is a relative date, the value is normalized to seconds and the integer seconds value is returned, if the argument is an absolute date, the duration in seconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of seconds from the present time to the future date is returned, if an absolute date in the past is passed as an argument, also 0 is returned (the function does not calculate second differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Example:

```
my int $us = get_duration_seconds(PT2M15S3u); # returns 135
```

Note

- equivalent to Qore::zzz8datezzz9::durationSeconds()
- to get the literal seconds integer value from a date/time value without calculating a duration, use get_seconds(date)

See also

- get_duration_milliseconds()
- get_duration_microseconds()

43.59.5.25 int Qore::get_epoch_seconds (date *dt*)

Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.

Code Flags:

CONSTANT

Parameters

dt	The date to process

Returns

the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch

Example:

```
my int $i = get_epoch_seconds(2012-01-19T08:02:15+01:00);
```

See also

timegm(date)

Note

This function is equivalent to mktime(date)

43.59.5.26 nothing Qore::get_epoch_seconds ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.27 int Qore::get_hours (date *dt*)

Returns an integer corresponding to the literal hour value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the hour value in the date

Example:

```
my int $n = get_hours($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::hours()

43.59.5.28 nothing Qore::get_hours ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.29 int Qore::get_microseconds (date *dt*)

Returns an integer corresponding to the literal microsecond value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the literal microsecond value in the date (does not calculate a duration)

Example:

```
my int $n = get_microseconds($dt);
```

Note

- equivalent to pseudo-method Qore::zzz8datezzz9::microseconds()
- to get the number of microseconds of duration in a date/time value, use get_duration_microseconds() instead

43.59.5.30 date Qore::get_midnight (date *dt*)

Returns midnight on the date passed (strips the time component on the new value)

Code Flags:

CONSTANT

Parameters

dt	the date to process

Returns

midnight on the date passed (strips the time component on the new value)

Example:

```
my date $midnight = get_midnight($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::midnight()

43.59.5.31 nothing Qore::get_midnight ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.32 int Qore::get_milliseconds (date *dt*)

Returns an integer corresponding to the literal millisecond value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the literal millisecond value in the date (does not calculate a duration)

Example:

```
my int $n = get_milliseconds($dt);
```

Note

- equivalent to pseudo-method Qore::zzz8datezzz9::milliseconds()
- to get the number of milliseconds of duration in a date/time value, use get_duration_milliseconds() instead

43.59.5.33 nothing Qore::get_milliseconds ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.34 int Qore::get_minutes (date *dt*)

Returns an integer corresponding to the literal minute value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the minute value in the date

Example:

```
my int $n = get_minutes($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::minutes()

43.59.5.35 nothing Qore::get_minutes ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.36 int Qore::get_months (date *dt*)

Returns an integer corresponding to the literal month value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the month value in the date

Example:

```
my int $n = get_months($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::months()

43.59.5.37 nothing Qore::get_months ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.38 int Qore::get_seconds (date *dt*)

Returns an integer corresponding to the literal second value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the literal second value in the date (does not calculate a duration)

Example:

```
my int $n = get_seconds($dt);
```

Note

- equivalent to pseudo-method Qore::zzz8datezzz9::seconds()
- to get the number of seconds of duration in a date/time value, use get_duration_seconds() instead

43.59.5.39 nothing Qore::get_seconds ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.40 int Qore::get_years (date *dt*)

Returns an integer corresponding to the literal year value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Parameters

dt	the date value; can be either a relative or absolute date

Returns

an integer corresponding to the year value in the date

Example:

```
my int $n = get_years($dt);
```

Note

equivalent to pseudo-method Qore::zzz8datezzz9::years()

43.59.5.41 nothing Qore::get_years ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.42 date Qore::getDateFromISOWeek (softint *year*, softint *week*, softint *day* = 1)

Returns an absolute date value for the ISO-8601 calendar week information passed (year, week number, optional: weekday, where 1=Monday, 7=Sunday) in the current time zone.

Throws an exception if the arguments are invalid

Code Flags:

RET_VALUE_ONLY

Parameters

year	the ISO-8601 year (which for some special days does not always correspond to the actual calendar year)
week	the ISO-8601 week number; if this argument is invalid, an ISO-8601-INVALID-WEEK exception is thrown
day	the day of the week, where 1=Monday and 7=Sunday; if this argument is invalid, an ISO-8601-INVALID-DAY exception is thrown

Returns

an absolute date value for the ISO-8601 calendar week information passed (year, week number, optional: weekday, where 1=Monday, 7=Sunday) in the current time zone

Example:

```
my date $d = getDateFromISOWeek(2007, 3); # returns 2007-01-15
```

Exceptions

ISO-8601-INVALID-WEEK	The week number is not valid for the given year
ISO-8601-INVALID-DAY	The day number is not between 1 (Monday) and 7 (Sunday) inclusive

43.59.5.43 int Qore::getDayNumber (date *dt*)

Returns an integer representing the ordinal day number in the year (corresponding to the ISO-8601 day number) for the absolute date value passed.

Code Flags:

CONSTANT

Parameters

dt	an absolute date value to get the ordinal day number

Returns

an integer representing the ordinal day number in the year (corresponding to the ISO-8601 day number) for the absolute date value passed; if a relative date value is passed, then this function will return 0

Example:

```
my int $dn = getDayNumber($dt);
```

43.59.5.44 nothing Qore::getDayNumber ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.45 int Qore::getDayOfWeek (date *dt*)

Returns an integer representing the day of the week for the absolute date value passed (0=Sunday, 6=Saturday)

Code Flags:

CONSTANT

Parameters

dt	an absolute date value to get the number for the day of the week

Returns

an integer representing the day of the week for the absolute date value passed (0=Sunday, 6=Saturday); if a relative date value is passed, then this function will return 0

Example:

```
my int $dn = getDayOfWeek($dt);
```

See also

getISODayOfWeek(date)

43.59.5.46 nothing Qore::getDayOfWeek ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.47 int Qore::getISODayOfWeek (date *dt*)

Returns an integer representing the ISO-8601 day of the week for the absolute date value passed (1=Monday, 7=Sunday)

Code Flags:

CONSTANT

Parameters

dt	an absolute date value to get the number for the day of the week

Returns

an integer representing the day of the week for the absolute date value passed (1=Monday, 7=Sunday); if a relative date value is passed, then this function will return 7

Example:

```
my int $dn = getISODayOfWeek($dt);
```

See also

getDayOfWeek(date)

43.59.5.48 nothing Qore::getISODayOfWeek ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.49 hash Qore::getISOWeekHash (date *dt*)

Returns a hash representing the ISO-8601 calendar week information for the absolute date passed (hash keys: `"year"`, `"week"`, `"day"`)

Note

that the ISO-8601 year does not always correspond with the calendar year at the end and the beginning of every year (for example 2006-01-01 in ISO-8601 calendar week format is: year=2005, week=52, day=7)

Code Flags:

CONSTANT

Parameters

dt	the date to get information for

Returns

a hash representing the ISO-8601 calendar week information for the absolute date passed (hash keys: "year", "week", "day"); if a relative date value is passed, then this function will return year=1970, week=1, day=1

Example:

```
my hash $h = getISOWeekHash(2007-05-15); # returns year=2007, week=20, day=2
```

See also

getISOWeekString(date)

43.59.5.50 nothing Qore::getISOWeekHash ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.51 string Qore::getISOWeekString (date *dt*)

Returns a string representing the ISO-8601 calendar week information for the absolute date passed (ex: 2006-01-01 = "2005-W52-7")

Code Flags:

CONSTANT

Parameters

dt	the date to get information for

Returns

a string representing the ISO-8601 calendar week information for the absolute date passed (ex: 2006-01-01 = "2005-W52-7"); if a relative date value is passed, then this function will return "1970-W01-1"

Example:

```
my string $str = getISOWeekString(2007-05-15); # returns "2007-W20-2"
```

See also

getISOWeekHash(date)

43.59.5.52 nothing Qore::getISOWeekString ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.53 date Qore::gmtime ( )

Returns the current UTC (GMT) time with a resolution of a second.

Returns

the current UTC (GMT) time with a resolution of a second

Code Flags:

CONSTANT

Example:

```
my date $utc = gmtime();
```

See also

- now_utc()
- localtime()

43.59.5.54 date Qore::gmtime (softint *secs*, softint *us* = 0)

Returns a date/time value in UTC (GMT) from arguments giving the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)

Code Flags:

CONSTANT

Parameters

secs	the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)
us	a microsecond offset for the time returned

Returns

a date/time value in UTC (GMT) from arguments giving the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)

Example:

```
my date $dt = gmtime(10);
```

See also

localtime(softint, softint)

43.59.5.55 date Qore::gmtime (date *dt*)

Returns the date and time in UTC (GMT) corresponding to the date argument passed.

Code Flags:

CONSTANT

Parameters

dt	a date to process; if the date passed is already in UTC, then the same value is returned

Returns

the date and time in UTC (GMT) corresponding to the date argument passed

Example:

```
my date $dt = gmtime(2012-01-19T15:00:00-07:00);
```

See also

localtime(date)

43.59.5.56 date Qore::hours (softint *hours*)

Returns a relative date/time value in hours based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

hours	the number of hours to return

Returns

a relative date/time value in hours based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = hours(100);
```

See also

- years(softint)
- months(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.57 nothing Qore::hours ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.58 bool Qore::is_date_absolute (date *dt*)

Returns True if the argument is an absolute date/time value, False if not.

Code Flags:

CONSTANT

Parameters

dt	the date to check

Returns

True if the argument is an absolute date/time value, False if not

Example:

```
my bool $b = is_date_absolute($dt)
```

See also

is_date_relative()

43.59.5.59 bool Qore::is_date_absolute ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.59.5.60 bool Qore::is_date_relative (date *dt*)

Returns True if the argument is an relative date/time value, False if not.

Code Flags:

CONSTANT

Parameters

dt	the date to check

Returns

True if the argument is an relative date/time value, False if not

Example:

```
my bool $b = is_date_relative($dt)
```

See also

is_date_absolute()

43.59.5.61 bool Qore::is_date_relative ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

NOOP

43.59.5.62 date Qore::localtime ( )

Returns the current date and time with a resolution to the second.

Returns

the current date and time with a resolution to the second

Code Flags:

CONSTANT

Example:

```
my date $d = localtime();
```

See also

gmtime()

Note

this variant of localtime() is equivalent to now()

43.59.5.63 date Qore::localtime (softint *secs*, softint *us* = 0)

Returns the date and time in the local time zone corresponding to the integer arguments passed, which are interpreted as the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)

Code Flags:

CONSTANT

Parameters

secs	the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)
us	a microsecond offset for the time returned

Returns

the date and time in the local time zone corresponding to the arguments passed

Example:

```
my date $dt = localtime(10);
```

See also

gmtime(softint, softint)

43.59.5.64 date Qore::localtime (date *dt*)

Returns the date and time in the local time zone corresponding to the date argument passed.

Code Flags:

CONSTANT

Parameters

dt	a date to process; if the date passed is in the local time zone, then the same value is returned

Returns

the date and time in the local time zone corresponding to the date argument passed

Example:

```
my date $dt = localtime(2012-01-19T15:00:00-07:00);
```

See also

gmtime(date)

43.59.5.65 date Qore::microseconds (softint *us*)

Returns a relative date/time value in microseconds based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

us	the number of microseconds to return

Returns

a relative date/time value in microseconds based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = microseconds(100);
```

See also

- years(softint)
- months(softint)
- days(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- milliseconds(softint)

43.59.5.66 date Qore::milliseconds (softint *ms*)

Returns a relative date/time value in milliseconds based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

ms	the number of milliseconds to return

Returns

a relative date/time value in milliseconds based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = milliseconds(100);
```

See also

- years(softint)
- months(softint)
- days(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- microseconds(softint)

43.59.5.67 nothing Qore::milliseconds ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.68 date Qore::minutes (softint *minutes*)

Returns a relative date/time value in minutes based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

minutes	the number of minutes to return

Returns

a relative date/time value in minutes based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = minutes(100);
```

See also

- years(softint)
- months(softint)
- days(softint)
- hours(softint)
- seconds(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.69 nothing Qore::minutes ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.70 int Qore::mktime (date *dt*)

Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.

Code Flags:

CONSTANT

Parameters

dt	The date to process

Returns

the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch

Example:

```
my int $i = mktime(2012-01-19T08:02:15+01:00);
```

See also

timegm(date)

Note

This function is equivalent to get_epoch_seconds(date)

43.59.5.71 nothing Qore::mktime ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.72 date Qore::months (softint *months*)

Returns a relative date/time value in months based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

months	the number of months to return

Returns

a relative date/time value in months based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = months(100);
```

See also

- years(softint)
- days(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.73 nothing Qore::months ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.74 date Qore::now ( )

Returns the current date and time with a resolution to the second.

Returns

the current date and time with a resolution to the second

Code Flags:

CONSTANT

Example:

```
my date $d = now();
```

See also

- now_ms()
- now_us()
- now_utc()

Note

this function is equivalent to localtime()

43.59.5.75 date Qore::now_ms ( )

Returns the current date and time with a resolution to the millisecond.

Returns

the current date and time with a resolution to the millisecond

Code Flags:

CONSTANT

Example:

```
my date $d = now_ms();
```

See also

- now() For a similar function returning the current date and time in the local time zone with coarser granularity, when resolution only to the second is needed
- now_us() for a similar function returning the current date and time with a resolution to the microsecond
- now_utc();

Note

There is no performance penalty for using now_ms() and now_us() versus now(); now() and now_ms() are kept for backwards compatibility

43.59.5.76 date Qore::now_us ( )

Returns the current date and time with a resolution to the microsecond.

Returns

the current date and time with a resolution to the microsecond

Code Flags:

CONSTANT

Example:

```
my date $d = now_us();
```

See also

- now() For a similar function returning the current date and time in the local time zone with coarser granularity, when resolution only to the second is needed
- now_ms() for a similar function returning the current date and time in the local time zone with coarser granularity, when resolution only to the millisecond is needed
- now_utc();

Note

There is no performance penalty for using now_ms() and now_us() versus now(); now() and now_ms() are kept for backwards compatibility

43.59.5.77 date Qore::now_utc ( )

Returns the current UTC date and time with a resolution to the microsecond.

Returns

the current UTC date and time with a resolution to the microsecond

Code Flags:

CONSTANT

Example:

```
my date $d = now_utc();
```

See also

now_us() for a similar function that returns the current date and time in the local time zone

43.59.5.78 date Qore::seconds (softint *seconds*)

Returns a relative date/time value in seconds based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

seconds	the number of seconds to return

Returns

a relative date/time value in seconds based on the integer argument passed to be used in date arithmetic

Example:

```
my date $rd = seconds(100);
```

See also

- years(softint)
- months(softint)
- days(softint)
- hours(softint)
- minutes(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.79 nothing Qore::seconds ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.80 int Qore::timegm (date *dt*)

Returns the number of seconds since January 1, 1970 00:00:00 in the local time zone for the given date.

Code Flags:

CONSTANT

Parameters

dt	The date to process

Returns

the number of seconds since January 1, 1970 00:00:00 in the local time zone for the given date

Example:

```
int secs = timegm($dt);
```

See also

get_epoch_seconds(date)

43.59.5.81 nothing Qore::timegm ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.59.5.82 date Qore::years (softint *years*)

Returns a relative date/time value in years based on the integer argument passed to be used in date arithmetic.

Code Flags:

CONSTANT

Parameters

years	the number of years to return

Returns

a relative date/time value in years based on the integer argument passed to be used in date arithmetic

Example:

```
my date $y = years(100);
```

See also

- months(softint)
- days(softint)
- hours(softint)
- minutes(softint)
- seconds(softint)
- milliseconds(softint)
- microseconds(softint)

43.59.5.83 nothing Qore::years ( )

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

43.60 Type Conversion Functions

Functions

- binary Qore::binary ()

  Always returns an empty binary object (of zero length)
- binary Qore::binary (null x)

  Always returns an empty binary object (of zero length)
- binary Qore::binary (softstring str)

  Returns a binary data type of the string passed; data types other than string will first be converted to a string and then returned as binary data.
- binary Qore::binary (binary bin)

  Always returns the same binary object passed.
- string Qore::binary_to_string (binary b, __7_ string encoding)

  Returns a string created from the binary data passed, taking an optional second argument giving the string encoding; if no second argument is passed then the default character encoding is assumed.
- bool Qore::boolean (any arg)

  Converts the argument to a boolean value.
- float Qore::float (softfloat f)

  Converts the argument to a floating-point (float) value.
- float Qore::float ()

  Always returns 0.0.
- hash Qore::hash (object obj)

  Returns a hash of an object's members.
- hash Qore::hash (list l)

  Returns a hash by taking even numbered list elements (starting with 0) and converting them to strings for the hash keys, and the odd numbered elements following the keys as the key value.
- hash Qore::hash (list keys, list values)

  Returns a hash by taking the first list as a list of keys, and the second list as a list of values.
- hash Qore::hash (hash h)

  Returns itself.
- hash Qore::hash ()

  Always returns the same hash passed.
- int Qore::int (string str, softint base)

  Converts the argument to an integer value.
- int Qore::int (softint i)

  Converts the argument to an integer value.
- int Qore::int ()

  Always returns 0.
- list Qore::list (...)

  Returns a list of the arguments passed at the top level.
- number Qore::number (softnumber n)

  Converts the argument to a number value.
- number Qore::number ()

  Always returns 0.0.
- string Qore::string (softstring str)

  Converts the argument to a string.
- string Qore::string ()

  Always returns an empty string.
- string Qore::type (any arg)

  Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

- string Qore::typename (any arg)

  Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

43.60.1 Detailed Description

43.60.2 Function Documentation

43.60.2.1 binary Qore::binary ( )

Always returns an empty binary object (of zero length)

Code Flags:

CONSTANT

The binary() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

- binary(softstring)
- binary(null)

43.60.2.2 binary Qore::binary (null *x*)

Always returns an empty binary object (of zero length)

Code Flags:

CONSTANT

The binary() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

binary(softstring)

43.60.2.3 binary Qore::binary (softstring *str*)

Returns a binary data type of the string passed; data types other than string will first be converted to a string and then returned as binary data.

This function is useful if, for example, a string type actually contains binary data; using this function will ensure that all data in the string (even if it contains embedded nulls) is maintained in the binary object (Qore strings must normally be terminated by a single null, so some Qore string operations do not work on binary data with embedded nulls).

Code Flags:

CONSTANT

Example:

```
my binary $b = binary($str);
```

43.60.2.4 binary Qore::binary (binary *bin*)

Always returns the same binary object passed.

Code Flags:

CONSTANT

The binary() function is used for type conversions, therefore this variant is not tagged with NOOP

43.60.2.5 string Qore::binary_to_string (binary *b*, __7_ string *encoding*)

Returns a string created from the binary data passed, taking an optional second argument giving the string encoding; if no second argument is passed then the default character encoding is assumed.

Code Flags:

CONSTANT

Parameters

b	the binary object to convert directly to a string
encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string created from the binary data passed

Example:

```
my string $str = binary_to_string($b, "iso-8859-1");
```

43.60.2.6 bool Qore::boolean (any *arg*)

Converts the argument to a boolean value.

Code Flags:

CONSTANT

Parameters

arg	the argument to convert to a boolean

Returns

the boolean value corresponding to the argument

Example:

```
my bool $b = boolean(1); # returns True
```

Note

this function behaves differently when %strict-bool-eval is set

43.60.2.7 float Qore::float (softfloat *f*)

Converts the argument to a floating-point (float) value.

Code Flags:

CONSTANT

Parameters

f	the argument to convert to a float

Returns

the float value corresponding to the argument

Example:

```
my float $i = float("3.1415");
```

43.60.2.8 float Qore::float ( )

Always returns 0.0.

Code Flags:

CONSTANT

The float() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

float(softfloat)

43.60.2.9 hash Qore::hash (object *obj*)

Returns a hash of an object's members.

Note

that if this function is called from outside the class' scope, the hash will only contain the object's public members

Code Flags:

RET_VALUE_ONLY

Parameters

obj	the object to process

Returns

a hash of an object's members

Example:

```
my hash $h = hash($obj);
```

43.60.2.10 hash Qore::hash (list *l*)

Returns a hash by taking even numbered list elements (starting with 0) and converting them to strings for the hash keys, and the odd numbered elements following the keys as the key value.

Code Flags:

RET_VALUE_ONLY

Parameters

l	the list to process in a manner similar to perl's hash initialization

Returns

a hash by taking even numbered list elements (starting with 0) and converting them to strings for the hash keys, and the odd numbered elements following the keys as the key value

Example:

```
my hash $h = hash(("a", 1, "b", "two"));
```

43.60.2.11 hash Qore::hash (list *keys*, list *values*)

Returns a hash by taking the first list as a list of keys, and the second list as a list of values.

If the two lists are of unequal sizes, then the keys list takes precedence (if the values list is longer, excess values are ignored, if the keys list is longer, then excess elements are assigned NOTHING)

Code Flags:

RET_VALUE_ONLY

Parameters

keys	a list of key values for the hash
values	a list of values for the hash, corresponding to the elements in *keys*

Returns

a hash by taking the first list as a list of keys, and the second list as a list of values

Example:

```
my hash $h = list($keys, $values);
```

43.60.2.12 hash Qore::hash (hash *h*)

Returns itself.

This function is used as an initializer and type converter, so this identity-variant is provided for consistency's sake

Code Flags:

CONSTANT

Parameters

h	the has to return

Returns

the hash passed

Example:

```
my hash $h = hash($h);
```

43.60.2.13 hash Qore::hash ( )

Always returns the same hash passed.

Code Flags:

CONSTANT

The hash() function is used for type conversions, therefore this variant is not tagged with NOOP

43.60.2.14 int Qore::int (string *str*, softint *base*)

Converts the argument to an integer value.

Code Flags:

RET_VALUE_ONLY

Parameters

str	the string to convert to an integer
base	the base of the integer in the string; this value must be 0 or 2 - 36 inclusive or an `INVALID-BASE` exception will be thrown

Returns

the integer value corresponding to the arguments

Example:

```
my int $i = int("fabc9", 16);
```

Exceptions

INVALID-BASE	the base is invalid; must be 0 or 2 - 36 inclusive

Since

Qore 0.8.5, this variant with the *base* argument is supported

43.60.2.15 int Qore::int (softint *i*)

Converts the argument to an integer value.

Code Flags:

CONSTANT

Parameters

i	the argument to convert to an integer

Returns

the integer value corresponding to the argument

Example:

```
my int $i = int("1");
```

43.60.2.16 int Qore::int ( )

Always returns 0.

Code Flags:

CONSTANT

The int() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

int(softint)

43.60.2.17 list Qore::list (...)

Returns a list of the arguments passed at the top level.

Code Flags:

CONSTANT

Parameters

...	the arguments that will be returned as a list

Returns

a list of the arguments passed at the top level; if a sole argument of NOTHING is passed, then an empty list is returned

Example:

```
my list $l = list(1, 2, 3, 4);
```

43.60.2.18 number Qore::number (softnumber *n*)

Converts the argument to a number value.

Code Flags:

CONSTANT

Parameters

n	the argument to convert to a number

Returns

the number value corresponding to the argument

Example:

```
my number $n = number("2.23040945718005e35");
```

43.60.2.19 number Qore::number ( )

Always returns 0.0.

Code Flags:

CONSTANT

The number() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

number(softnumber)

43.60.2.20 string Qore::string (softstring *str*)

Converts the argument to a string.

Code Flags:

CONSTANT

Parameters

str	the argument to convert to a string

Returns

the string value corresponding to the argument

Example:

```
my string $str = string(100);
```

43.60.2.21 string Qore::string ( )

Always returns an empty string.

Code Flags:

CONSTANT

The string() function is used for type conversions, therefore this variant is not tagged with NOOP

See also

string(softstring)

43.60.2.22 string Qore::type (any *arg*)

Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

Code Flags:

CONSTANT

Parameters

arg	the argument to check

Returns

a string giving the data type of the argument passed; see String Type Constants for the values returned by this function

Example:

```
my string $type = type($v);
```

Note

- This function is identical to typename(any) and to calling pseudo-method Qore::zzz8valuezzz9::type()
- It is faster and more efficient to use Qore::zzz8valuezzz9::typeCode() for comparing data types

See also

pseudo-method Qore::zzz8valuezzz9::typeCode()

43.60.2.23 string Qore::typename (any *arg*)

Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

Code Flags:

CONSTANT

Parameters

arg	the argument to check

Returns

a string giving the data type of the argument passed; see String Type Constants for the values returned by this function

Example:

```
my string $type = typename($v);
```

Note

- This function is identical to type(any) and to calling pseudo-method Qore::zzz8valuezzz9::type()
- It is faster and more efficient to use Qore::zzz8valuezzz9::typeCode() for comparing data types

43.61 String Type Constants

Variables

- const Qore::Type::Binary = "binary"

  Gives the type for binary values.
- const Qore::Type::Boolean = "bool"

  Gives the type for boolean values.
- const Qore::Type::CallReference = "call reference"

  Gives the type for call references.
- const Qore::Type::Closure = "closure"

  Gives the type for closures.
- const Qore::Type::Date = "date"

  Gives the type for the date values.
- const Qore::Type::Float = "float"

  Gives the type for float values.
- const Qore::Type::Hash = "hash"

  Gives the type for hash values.
- const Qore::Type::Int = "integer"

  Gives the type for integer values.
- const Qore::Type::List = "list"

  Gives the type for list values.
- const Qore::Type::NothingType = "nothing"

  Gives the type when no value is available.
- const Qore::Type::NullType = "NULL"

  Gives the type for SQL null values.
- const Qore::Type::Number = "number"

  Gives the type for number values.
- const Qore::Type::Object = "object"

  Gives the type for object values.
- const Qore::Type::String = "string"

  Gives the type for string values.

43.61.1 Detailed Description

String type constants as returned by type() and typename()

Chapter 44

Namespace Documentation

44.1 Qore Namespace Reference

main Qore-language namespace

Namespaces

- Err

  Qore::Err namespace.
- Option

  Qore::Option namespace.
- SQL

  Qore::SQL namespace.
- Thread

  Qore::Thread namespace.
- Type

  Qore::Type namespace.

Classes

- class AbstractBidirectionalIterator

  This class defines an abstract interface for bidirectional iterators.
- class AbstractIterator

  This class defines an abstract interface for iterators.
- class AbstractQuantifiedBidirectionalIterator

  This class defines an abstract interface for bidirectional iterators where the size of the object is known in advance.
- class AbstractQuantifiedIterator

  This class defines an abstract interface for iterators where the size of the object being iterated is known in advance.
- class Dir

  This class implements directory handling, file listing, creating/removing subdirectories, etc.
- class File

  The File class allows Qore programs to read, write, and create files.
- class FileLineIterator

  This class defines a line iterator for text files.
- class FtpClient

  The FtpClient class allows Qore code to communicate with FTP servers with the FTP and FTPS protocols.
- class GetOpt

The GetOpt class provides an easy way to process POSIX-style command-line options in Qore scripts/programs.

- class HashIterator

  This class an iterator class for hashes.

- class HashKeyIterator

  This class an iterator class for hashes.

- class HashKeyReverseIterator

  This class an iterator class for hashes.

- class HashListIterator

  This class an iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

- class HashListReverseIterator

  This class a reverse iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

- class HashPairIterator

  This class an iterator class for hashes.

- class HashPairReverseIterator

  This class an iterator class for hashes.

- class HashReverseIterator

  This class an iterator class for hashes.

- class HTTPClient

  The HTTPClient class can be used to communicate with HTTP servers with and without TLS/SSL encryption.

- class ListHashIterator

  This class an iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return lists of columns where each list entry is a hash of the current column values.

- class ListHashReverseIterator

  This class a reverse iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return hashes with keys giving column names where the key values are lists of column values.

- class ListIterator

  This class an iterator class for lists.

- class ListReverseIterator

  This class an iterator class for lists.

- class ObjectIterator

  This class a basic iterator class for objects.

- class ObjectKeyIterator

  This class an iterator class for objectes.

- class ObjectKeyReverseIterator

  This class an iterator class for objects.

- class ObjectPairIterator

  This class an iterator class for objects.

- class ObjectPairReverseIterator

  This class an iterator class for objects.

- class ObjectReverseIterator

  This class an iterator class for objects.

- class Program

  Program objects allow Qore programs to support subprograms with the option to restrict capabilities, for example, to support user-defined logic for application actions.

- class RangeIterator

  This class defines a range-like iterator to be used to iterate numerical sequences.

- class ReadOnlyFile

The ReadOnlyFile class allows Qore programs to read existing files.

- class SingleValueIterator

  This class defines a simple iterator to be used to iterate single values (or complex objects where no iterator has been implemented yet)
- class Socket

  The Socket class allows Qore programs safe access to network sockets.
- class SSLCertificate

  SSLCertificate objects allow Qore code to work with X.509 certificate data.
- class SSLPrivateKey

  This class implements a container for private key data.
- class TermIOS

  This class allows Qore scripts to get or set terminal settings on UNIX platforms.
- class TimeZone

  The TimeZone class provides access to time zone functionality.
- class zzz8binaryzzz9

  Methods in this pseudo-class can be executed on binary values.
- class zzz8boolzzz9

  Methods in this pseudo-class can be executed on booling-point values.
- class zzz8callrefzzz9

  Methods in this pseudo-class can be executed on call references.
- class zzz8closurezzz9

  Methods in this pseudo-class can be executed on closures.
- class zzz8datezzz9

  Methods in this pseudo-class can be executed on date/time value types.
- class zzz8floatzzz9

  Methods in this pseudo-class can be executed on floating-point values.
- class zzz8hashzzz9

  Methods in this pseudo-class can be executed on hash values.
- class zzz8intzzz9

  Methods in this pseudo-class can be executed on integer values.
- class zzz8listzzz9

  Methods in this pseudo-class can be executed on lists.
- class zzz8nothingzzz9

  Methods in this pseudo-class can be executed on NOTHING.
- class zzz8numberzzz9

  Methods in this pseudo-class can be executed on arbitrary precision number values.
- class zzz8objectzzz9

  Methods in this pseudo-class can be executed on objects.
- class zzz8stringzzz9

  Methods in this pseudo-class can be executed on strings.
- class zzz8valuezzz9

  Methods in this pseudo-class are available to be executed on any value type (even NOTHING); this is the root class for all pseudo-classes.

Functions

- RangeIterator xrange (int start, int stop, int step=1)

  Returns a RangeIterator containing an arithmetic progression of integers.
- RangeIterator xrange (int stop)

  Returns a RangeIterator containing an arithmetic progression of integers with start = 0 and step = 1.
- binary bunzip2_to_binary (binary bin)

Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a binary object.

- nothing bunzip2_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string bunzip2_to_string (binary bin, __7_ string encoding)

  Uncompresses the given data with the bzip2 algorithm and returns the uncompressed data as a string.

- nothing bunzip2_to_string ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary bzip2 (binary bin, softint level=BZ2_DEFAULT_COMPRESSION)

  Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.

- binary bzip2 (string str, softint level=BZ2_DEFAULT_COMPRESSION)

  Compresses the given data with the bzip2 algorithm and returns the compressed data as a binary.

- nothing bzip2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary compress (string str, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.

- binary compress (binary bin, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "deflate" data compression (RFC 1951) and returns a binary object of the compressed data.

- nothing compress ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary gunzip_to_binary (binary bin)

  Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a binary object of the uncompressed data.

- nothing gunzip_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string gunzip_to_string (binary bin, __7_ string encoding)

  Performs zlib-based decompression of data compressed with the "gzip" algorithm (RFC 1952) and returns a string of the uncompressed datas.

- nothing gunzip_to_string ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary gzip (string str, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.

- binary gzip (binary bin, int level=Z_DEFAULT_COMPRESSION)

  Performs zlib-based "gzip" data compression (RFC 1952) and returns a binary object of the compressed data.

- nothing gzip ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary uncompress_to_binary (binary bin)

  Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a binary object of the decompressed data.

- nothing uncompress_to_binary ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string uncompress_to_string (binary bin, __7_ string encoding)

  Performs zlib-based decompression of data compressed by the "deflate" algorithm (RFC 1951) and returns a string of the decompressed data.

- nothing uncompress_to_string ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool cx_first ()

  Returns True if currently iterating the first element in a context statement, False if not.

- bool cx_last ()

  Returns True if currently iterating the last element in a context statement, False if not.

- int cx_pos ()

  Returns the current row number within the active context statement (starting from 0)

- int cx_total ()

  Returns the total number of rows within the active context statement.

- any cx_value (string key)

  Returns the current value of the given column while iterating a context statement.

- binary blowfish_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for the blowfish algorithm and returns a binary object of the decrypted data.

- string blowfish_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for the blowfish algorithm and returns a string of the decrypted data.

- binary blowfish_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for the blowfish algorithm

- binary cast5_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

- string cast5_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

- binary cast5_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for the CAST5 algorithm using a variable-length key and an optional 8-byte initialization vector.

- binary des_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

- string des_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

- binary des_ede3_decrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

- string des_ede3_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

- binary des_ede3_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for the three-key triple DES algorithm using three 8-byte keys (set by a single 24-byte key argument) and an optional 8-byte initialization vector.

- binary des_ede_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

- string des_ede_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

- binary des_ede_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for the two-key triple DES algorithm using two eight-byte keys (set by a single 16-byte key argument)

- binary des_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for the DES algorithm using an 8-byte key.

- binary des_random_key ()

  Returns a binary object of a random key for the DES algorithm
- binary desx_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.
- string desx_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.
- binary desx_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for RSA's DESX algorithm using a 24-byte key and an optional 8-byte initialization vector.
- binary rc2_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- string rc2_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- binary rc2_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- binary rc4_decrypt (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- string rc4_decrypt_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- binary rc4_encrypt (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Alleged RC4 cipher algorithm, which should be compatible with RSA's RC4(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- binary rc5_decrypt_cbc (binary data, data key, data iv=Qore::DefaultIV)

  Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- string rc5_decrypt_cbc_to_string (binary data, data key, data iv=Qore::DefaultIV, __7_ string encoding)

  Decrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- binary rc5_encrypt_cbc (data data, data key, data iv=Qore::DefaultIV)

  Encrypts data using the Cipher Block Chaining function for RSA's RC2(tm) algorithm using a variable-length key and an optional 8-byte initialization vector.
- string DSS (data data)

  Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a hex string.
- string DSS1 (data data)

  Returns the DSS1 message digest (based on SHA1 and DSA) of the supplied argument as a hex string.
- binary DSS1_bin (data data)

  Returns the DSS1 message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.
- binary DSS_bin (data data)

  Returns the DSS message digest (based on SHA-0 and DSA) of the supplied argument as a binary object.
- string MD2 (data data)

  Returns the MD2 message digest of the supplied argument as a hex string.
- binary MD2_bin (data data)

  Returns the MD2 message digest of the supplied argument as binary object.
- string MD4 (data data)

  Returns the MD4 message digest of the supplied argument as a hex string.
- binary MD4_bin (data data)

Returns the MD4 message digest of the supplied argument as a binary object.

- string MD5 (data data)

  Returns the MD5 message digest of the supplied argument as a hex string.
- binary MD5_bin (data data)

  Returns the MD5 message digest of the supplied argument as a binary object.
- string MDC2 (data data)

  Returns the MDC2 message digest of the supplied argument as a hex string.
- binary MDC2_bin (data data)

  Returns the MDC2 message digest of the supplied argument as a binary object.
- string RIPEMD160 (data data)

  Returns the RIPEMD message digest of the supplied argument as a hex string.
- binary RIPEMD160_binary (data data)

  Returns the RIPEMD message digest of the supplied argument as a binary object.
- string SHA (data data)

  Returns the SHA (outdated SHA-0) message digest of the supplied argument as a hex string.
- string SHA1 (data data)

  Returns the SHA1 message digest of the supplied argument as a hex string.
- binary SHA1_bin (data data)

  Returns the SHA1 message digest of the supplied argument as a binary object.
- string SHA224 (data data)

  Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary SHA224_bin (data data)

  Returns the SHA-224 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string SHA256 (data data)

  Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary SHA256_bin (data data)

  Returns the SHA-256 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string SHA384 (data data)

  Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary SHA384_bin (data data)

  Returns the SHA-384 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- string SHA512 (data data)

  Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a hex string.
- binary SHA512_bin (data data)

  Returns the SHA-512 message digest (a variant of SHA-2) of the supplied argument as a binary object.
- binary SHA_bin (data data)

  Returns the SHA (outdated SHA-0) message digest of the supplied argument as a binary object.
- string DSS1_hmac (data data, string key)

  Returns the DSS1 (SHA-1 and DSA) based HMAC of the supplied argument as a hex string.
- string DSS_hmac (data data, string key)

  Returns the DSS (SHA-0 and DSA) based HMAC of the supplied argument as a hex string.
- string MD2_hmac (data data, string key)

  Returns the MD2 based HMAC of the supplied argument as a hex string.
- string MD4_hmac (data data, string key)

  Returns the MD4 based HMAC of the supplied argument as a hex string.
- string MD5_hmac (data data, string key)

  Returns the MD5 based HMAC of the supplied argument as a hex string.
- string MDC2_hmac (data data, string key)

  Returns the MDC2 based HMAC of the supplied argument as a hex string.
- string RIPEMD160_hmac (data data, string key)

  Returns the RIPEMD based HMAC of the supplied argument as a hex string.

- string SHA1_hmac (data data, string key)

  Returns the SHA1 based HMAC of the supplied argument as a hex string.
- string SHA224_hmac (data data, string key)

  Returns the SHA224 based HMAC of the supplied argument as a hex string.
- string SHA256_hmac (data data, string key)

  Returns the SHA256 based HMAC of the supplied argument as a hex string.
- string SHA384_hmac (data data, string key)

  Returns the SHA384 based HMAC of the supplied argument as a hex string.
- string SHA512_hmac (data data, string key)

  Returns the SHA512 based HMAC of the supplied argument as a hex string.
- string SHA_hmac (data data, string key)

  Returns the SHA based HMAC of the supplied argument as a hex string.
- __7_ string getenv (string var)

  Retrieves the value of an environment variable or NOTHING if the variable is not set.
- nothing getenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int setenv (string env, softstring val)

  Sets an environment variable to a value.
- nothing setenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int unsetenv (string env)

  Unsets an environment variable.
- nothing unsetenv ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int chdir (string path)

  Changes the current working directory for the current process.
- int chmod (string path, softint mode)

  Changes the mode of a file or directory.
- int chown (string path, softint owner=-1, softint group=-1)

  Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), follows symbolic links.
- string getcwd ()

  Returns a string giving the current working directory or NOTHING if the current working directory could not be read.
- string getcwd2 ()

  Returns a string giving the current working directory; throws an exception if the current directory cannot be read.
- __7_ list glob (string glob_str)

  Returns a list of files matching the string argument or NOTHING if the call to glob() fails.
- nothing glob ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash hlstat (string path)

  Returns a hash of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.
- nothing hlstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash hstat (string path)

  Returns a hash of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

- nothing hstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool is_bdev (string path)

  Returns True if the string passed identifies a block device on the filesystem, False if not.
- bool is_cdev (string path)

  Returns True if the string passed identifies a character device on the filesystem, False if not.
- bool is_dev (string path)

  Returns True if the string passed identifies a device (either block or character) on the filesystem, False if not.
- bool is_dir (string path)

  Returns True if the string passed identifies a directory on the filesystem, False if not.
- bool is_executable (string path)

  Returns True if the string passed identifies an executable on the filesystem, False if not.
- bool is_file (string path)

  Returns True if the string passed identifies a regular file on the filesystem, False if not.
- bool is_link (string path)

  Returns True if the string passed identifies a symbolic link on the filesystem, False if not.
- bool is_pipe (string path)

  Returns True if the string passed identifies a pipe (FIFO) on the filesystem, False if not.
- bool is_readable (string path)

  Returns True if the string passed identifies a file readable by the current user, False if not.
- bool is_socket (string path)

  Returns True if the string passed identifies a socket on the filesystem, False if not.
- bool is_writable (string path)

  Returns True if the string passed identifies a file writable by the current user, False if not.
- bool is_writeable (string path)

  Returns True if the string passed identifies a file writable by the current user (backwards-compatible misspelling of is_writable())
- int lchown (string path, softint uid=-1, softint gid=-1)

  Changes the user and group owners of a file, if the current user has permission to do so (normally only the superuser can change the user owner), does not follow symbolic links but rather operates on the symbolic link itself.
- __7_ list lstat (string path)

  Returns a list of file status values for the path argument and does not follow symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.
- nothing lstat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int mkdir (string path, softint mode=0777)

  Creates a directory, optionally specifying the mode.
- int mkfifo (string path, softint mode=0600)

  Creates a named pipe file with an optional file mode.
- string readlink (string path)

  Returns the target of a symbolic link; throws an exception if an error occurs (ex: file does not exist or is not a symbolic link)
- nothing rename (string old_path, string new_path)

  Renames (or moves) files or directories. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem objects or paths to execute the rename operation.
- int rmdir (string path)

  Removes a directory.
- __7_ list stat (string path)

  Returns a list of file status values for the path argument, following any symbolic links; if any errors occur, NOTHING is returned and errno() can be used to retrieve the error number.

- nothing stat ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash statvfs (string path)

  Returns a hash of filesystem status values for the file or directory path passed.
- nothing symlink (string old_path, string new_path)

  Creates a symbolic link to a directory path. Note that for this call to function properly, the Qore process must have sufficient permissions and access to the given filesystem path to create the symbolic link.
- int umask (softint mask)

  Sets the file creation mode mask for the process and returns the previous value of the file creation mode mask.
- nothing umask ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int unlink (string path)

  Deletes a file and returns 0 for success, -1 for error (in which case errno() can be used to get the error)
- nothing unlink ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing abort ()

  Aborts the current program (this function does not return)
- string basename (string path)

  Returns a string giving the last element of a file path (meant to be the filename)
- nothing basename ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing close_all_fd (__7_ softbool strd)

  closes all possible file descriptors; useful in "daemon" processes that may have inherited open file descriptors
- string dirname (string path)

  Returns a string giving the path up to a file but not the filename itself.
- nothing dirname ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int errno ()

  Returns the error code of the last error that occurred in the current thread.
- nothing exec (string command)

  Replaces the current process image with another; this function does not return.
- nothing exit (softint rc=0)

  Exits the program with the return code passed (this function does not return)
- int fork ()

  Creates a copy of the current process with a new PID; returns 0 in the child process; returns the child's PID in the parent process.
- list getaddrinfo (__7_ string node, __7_ softstring service, softint family=AF_UNSPEC, softint flags=0)

  Returns a list of Address Information Hash for the given node name or string address; if no lookup can be performed then an exception is thrown.
- int getegid ()

  Returns the effective group ID of the current process.
- int geteuid ()

  Returns the effective user ID of the current process.
- int getgid ()

  Returns the real group ID of the current process.
- list getgroups ()

  returns a list of group IDs that the user is a member of

- __7_ string gethostbyaddr (string addr, softint type=AF_INET)

  Returns the official hostname corresponding to the network addressed passed as an argument.
- nothing gethostbyaddr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash gethostbyaddr_long (string addr, softint type=AF_INET)

  Returns a hash representing all host and address information corresponding to the address and address type passed as arguments.
- nothing gethostbyaddr_long ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ string gethostbyname (string name)

  Returns the first address corresponding to the hostname passed as an argument or NOTHING if the lookup fails.
- nothing gethostbyname ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ hash gethostbyname_long (string name)

  Returns a hash representing all host and address information corresponding to the hostname passed as an argument.
- nothing gethostbyname_long ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string gethostname ()

  Returns the hostname of the system.
- int getpid ()

  Returns the PID (process ID) of the current process.
- int getppid ()

  Returns the PID (process ID) of the parent process of the current process.
- int getuid ()

  Returns the real user ID of the current process.
- int kill (softint pid, softint sig=SIGHUP)

  Sends a signal to a process, if no signal number is given, then Qore::SIGHUP is sent by default.
- nothing kill ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int rand ()

  Returns a random 64-bit integer number.
- int setegid (softint gid)

  Changes the process effective group ID according to the argument passed.
- int seteuid (softint uid)

  Changes the effective process user ID according to the argument passed.
- int setgid (softint gid)

  Changes the process group ID according to the argument passed.
- setgroups (softlist gids)

  sets the list of supplementary group IDs for the current process
- int setsid ()

  Creates a new session lead by the calling process.
- int setuid (softint uid)

  Changes the process user ID according to the argument passed.
- int sleep (softint seconds)

  Causes the current thread to sleep for a certain number of seconds.
- nothing sleep ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- nothing srand (softint seed)

  Seeds the random number generator with the integer passed.

- nothing srand ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string strerror (softint err)

  Returns the string corresponding to the error code passed (generally retrieved with errno())

- string strerror ()

  Returns the string corresponding to the last error that occurred in the current thread.

- int system (string command)

  executes a command and returns the exit code of the process

- nothing system ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int usleep (softint usecs)

  Causes the current thread to sleep for a certain number of microseconds.

- int usleep (date d)

  Causes the current thread to sleep for a certain number of microseconds.

- nothing usleep ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool inlist ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool inlist (any arg, nothing x)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool inlist (any arg, softlist l)

  Returns True if the first argument is a member of the second argument list using soft comparisons (with implicit type conversions), False if not.

- bool inlist_hard ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool inlist_hard (any arg, nothing x)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool inlist_hard (any arg, softlist l)

  Returns True if the first argument is a member of the second argument list using hard comparisons (without any implicit type conversions), False if not.

- any max (list l)

  Returns the maximum value in a list.

- any max (list l, string func)

  Returns the maximum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.

- any max (list l, code f)

  Returns the maximum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.

- any max (...)

  Returns the maximum value of the arguments passed to the function.

- any min (list l)

  Returns the minumum value in a list.

- any min (list l, string func)

  Returns the minumum value in a list; accepts the name of a function to use to compare complex data types or to give a special sort order.
- any min (list l, code f)

  Returns the minumum value in a list; accepts a call reference or a closure to use to compare complex data types or to give a special sort order.
- any min (...)

  Returns the minumum value of the arguments passed to the function.
- list range (int start, int stop, int step=1)

  Returns a list containing an arithmetic progression of integers.
- list range (int stop)

  Returns a list containing an arithmetic progression of integers with start = 0 and step = 1.
- nothing reverse ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- list reverse (list l)

  Reverses a list and returns the new list.
- any sort (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- list sort (list l)

  Performs an unstable sort in ascending order and returns the new list.
- list sort (list l, string func)

  Performs an unstable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.
- list sort (list l, code f)

  Performs an unstable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.
- any sortDescending (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- list sortDescending (list l)

  Performs an unstable sort in descending order and returns the new list.
- list sortDescending (list l, string func)

  Performs an unstable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.
- list sortDescending (list l, code f)

  Performs an unstable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.
- any sortDescendingStable (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- list sortDescendingStable (list l)

  Performs a stable sort in descending order and returns the new list.
- list sortDescendingStable (list l, string func)

  Performs a stable sort in descending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.
- list sortDescendingStable (list l, code f)

  Performs a stable sort in descending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.
- any sortStable (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list sortStable (list l)

  Performs a stable sort in ascending order and returns the new list.
- list sortStable (list l, string func)

  Performs a stable sort in ascending order and returns the new list; accepts the name of a function to use to sort complex data types or to give a special sort order.
- list sortStable (list l, code f)

  Performs a stable sort in ascending order and returns the new list; accepts a call reference or a closure to use to sort complex data types or to give a special sort order.
- int abs (int i)

  Returns the absolute value of the argument passed.
- number abs (number n)

  Returns the absolute value of the argument passed.
- float abs (softfloat f)

  Returns the absolute value of the argument passed.
- float abs ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number acos (number n)

  Returns the value in radians of the arc cosine of the given value.
- float acos (softfloat f)

  Returns the value in radians of the arc cosine of the given value.
- number asin (number n)

  Returns the value in radians of the arc sine of the given value.
- float asin (softfloat f)

  Returns the value in radians of the arc sine of the given value.
- float asin ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number atan (number n)

  Returns the value in radians of the arc tangent of the given value.
- float atan (softfloat f)

  Returns the value in radians of the arc tangent of the given value.
- float atan ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number atan2 (number y, number x)

  Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.
- float atan2 (softfloat y, softfloat x)

  Returns the principal value of the arc tangent of y/x, using the signs of the two arguments to determine the quadrant of the result.
- float atan2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number cbrt (number n)

  Returns the cube root of the number passed.
- float cbrt (softfloat f)

  Returns the cube root of the number passed.
- float cbrt ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number ceil (number n)

Returns a number equal to the smallest integral value greater than or equal to the argument passed.

- float ceil (softfloat f)

  Returns a floating-point number equal to the smallest integral value greater than or equal to the argument passed.

- float ceil ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number cos (number n)

  Returns the cosine of the number in radians passed.

- float cos (float f)

  Returns the cosine of the number in radians passed.

- float cos ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number cosh (number n)

  Returns the hyperbolic cosine of the given value.

- float cosh (softfloat f)

  Returns the hyperbolic cosine of the given value.

- float cosh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number exp (number n)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number.

- float exp (softfloat f)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number.

- float exp ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number exp2 (number n)

  Returns the value of 2 raised to the power of the given number.

- float exp2 (softfloat f)

  Returns the value of 2 raised to the power of the given number.

- float exp2 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number expm1 (number n)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number - 1.

- float expm1 (softfloat f)

  Returns the value of e (the base of natural logarithms) raised to the power of the given number - 1.

- float expm1 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number floor (softnumber n)

  Returns a number equal to the largest integral value less than or equal to the argument passed.

- float floor (softfloat f)

  Returns a floating-point number equal to the largest integral value less than or equal to the argument passed.

- float floor ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number hypot (number x, number y)

  Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.

- float hypot (softfloat x, softfloat y)

  Returns the length of the hypotenuse of a right-angle triangle with sides given as the two arguments.

- float hypot ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number log10 (number n)

  Returns the base 10 logarithm of the given number.
- float log10 (softfloat f)

  Returns the base 10 logarithm of the given number.
- float log10 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number log1p (number n)

  Returns the natural logarithm of 1 + the given number.
- float log1p (softfloat f)

  Returns the natural logarithm of 1 + the given number.
- float log1p ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- float logb (softfloat f)

  Returns the exponent of the given number.
- float logb ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number nlog (number n)

  Returns the natural logarithm of the given value.
- float nlog (softfloat f)

  Returns the natural logarithm of the given value.
- float nlog ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number pow (number x, number y)

  Returns a number raised to the power of another number.
- float pow (softfloat x=0.0, softfloat y=0.0)

  Returns a number raised to the power of another number.
- number round (number n)

  Returns a number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.
- float round (softfloat f)

  Returns a floating-point number equal to the closest integer to the argument passed; numbers halfway between two integers are rounded away from zero.
- float round ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number sin (number n)

  Returns the sine of the number in radians passed.
- float sin (softfloat f)

  Returns the sine of the number in radians passed.
- float sin ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- number sinh (number n)

  Returns the hyperbolic sine of the given value.
- float sinh (softfloat f)

Returns the hyperbolic sine of the given value.

- float sinh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number sqrt (number n)

  Returns the square root of the number passed.

- float sqrt (softfloat f)

  Returns the square root of the number passed.

- float sqrt ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number tan (number n)

  Returns the tangent of the number in radians passed.

- float tan (softfloat f)

  Returns the tangent of the number in radians passed.

- float tan ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- number tanh (number n)

  Returns the hyperbolic tangent of the given value.

- float tanh (softfloat f)

  Returns the hyperbolic tangent of the given value.

- float tanh ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- nothing remove_signal_handler (softint signal)

  Removes a signal handler and returns the signal handling state to the default.

- nothing set_signal_handler (softint signal, code f)

  Sets or replaces a signal handler according to the signal number and closure or call reference (function or object method reference) passed.

- string backquote (string cmd, __7_ reference rc)

  Executes a process and returns a string of the output (stdout only)

- nothing backquote ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- any call_builtin_function (string name,...)

  Calls a function and returns the return value, passing the remaining arguments after the function name to the builtin function.

- any call_builtin_function_args (string name, __7_ softlist vargs)

  Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.

- any call_function (string name,...)

  Calls a function and returns the return value, passing the remaining arguments after the function name to the function.

- any call_function (code f,...)

  Calls the given call reference or closure and returns the result, passing the remaining arguments to the call reference or closure.

- any call_function_args (string name, __7_ softlist vargs)

  Calls a function and returns the return value, using the optional second argument as a list of arguments for the function.

- any call_function_args (code f, __7_ softlist vargs)

  Calls the given call reference or closure and returns the result, using the optional second argument as a list of arguments to the call reference or closure.

- string decode_url (string url)

Decodes percent numeric codes in a URL string and returns the decoded string in UTF-8 encoding.

- nothing decode_url ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string encode_url (string url, softbool encode_all=False)

  Encodes URLs by substituting '%' characters with '%25', spaces (' ') with '%20', and non-ascii characters by percent-encoded representations.

- bool exists (...)

  A function performing the same role as the exists operator.

- bool existsFunction (string name)

  Returns True if the function exists in the current program's function name space.

- bool existsFunction (code c)

  Always returns True.

- nothing existsFunction ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ string functionType (string name)

  Returns "builtin" (for a builtin function), "user" (for a user function), or NOTHING (if the function cannot be found) according to the function name passed.

- nothing functionType ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ int getByte (string str, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- nothing getByte ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ int getByte (binary b, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- string getClassName (object obj)

  Returns the class name of the object passed.

- nothing getClassName ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list getFeatureList ()

  Returns a list of strings of the builtin and module-supplied features of Qore.

- hash getModuleHash ()

  Returns a hash of hashes describing the currently-loaded Qore modules; the top-level hash keys are the module names.

- list getModuleList ()

  Returns a list of hashes describing the currently-loaded Qore modules.

- __7_ int getWord32 (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int getWord32 (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- nothing getWord32 ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ int get_byte (string str, softint offset=0)

Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_byte (binary b, softint offset=0)

  Returns the byte value at the given byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- string get_default_encoding ()

  Returns the name of the default character encoding.

- string get_ex_pos (hash ex)

  returns a descriptive string for an exception location; the `source` and `offset` information will also be included in the string returned if present in the exception hash argument

- int get_parse_options ()

  returns the current parse options for the current Program object

- hash get_qore_library_info ()

  Returns a hash of library build and version info.

- hash get_qore_option_hash ()

  Returns a hash of hashes giving information about Qore library options for the current build.

- list get_qore_option_list ()

  Returns a list of hashes giving information about Qore library options for the current build.

- __7_ string get_script_dir ()

  Returns the name of the directory from which the current script was executed or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ string get_script_name ()

  Returns the filename of the current script if known or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ string get_script_path ()

  Returns the path (directory and filename) of the current script or NOTHING if unknown (i.e. no parent script, script read from stdin, etc)

- __7_ int get_word_16 (string str, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_16 (binary b, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_16_lsb (string str, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_16_lsb (binary b, softint offset=0)

  Returns the 16-bit integer value at the given 2-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_32 (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_32 (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_32_lsb (string str, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_32_lsb (binary b, softint offset=0)

  Returns the 32-bit integer value at the given 4-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_64 (string str, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.

- __7_ int get_word_64 (binary b, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int get_word_64_lsb (string str, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- __7_ int get_word_64_lsb (binary b, softint offset=0)

  Returns the 64-bit integer value at the given 8-byte offset (the first value is at offset 0) or NOTHING if the offset is not legal for the given data.
- bool has_key (hash h, string key)

  Returns True if the given key exists in the hash (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered.
- bool has_key (object obj, string key)

  Returns True if the given key exists in the object (does not necessarily have to have a value assigned); exceptions are only raised if string encoding errors are encountered or in case of object access errors.
- list hash_values (hash h)

  Returns a list of all the values in the hash argument passed.
- nothing hash_values ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int hextoint (string str)

  Returns an integer for a hexadecimal string value; throws an exception if non-hex digits are found.
- nothing hextoint ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string html_decode (string str)

  Returns a string with any HTML escape codes translated to the original characters.
- nothing html_decode ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string html_encode (string str)

  Returns a string with characters needing HTML escaping translated to HTML escape codes.
- nothing html_encode ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing load_module (string name)

  Loads in a Qore module at run-time.
- nothing load_module ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string makeBase64String (string str, softint maxlinelen=-1)

  Returns a base64-encoded representation of a string.
- string makeBase64String (binary bin, softint maxlinelen=-1)

  Returns a base64-encoded representation of a binary object.
- nothing makeBase64String ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string makeHexString (string str)

  Returns a hex-encoded representation of a string.
- string makeHexString (binary bin)

  Returns a hex-encoded representation of a binary object.
- nothing makeHexString ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash parse (string code, string label, __7_ softint warning_mask, __7_ string source, __7_ softint offset, softbool format_label=True)

  Adds the text passed to the current program's code, tagged with the given label.

- nothing parse ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary parseBase64String (string str)

  Parses a base64 encoded string and returns a binary object of the decoded data.

- nothing parseBase64String ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string parseBase64StringToString (string str, __7_ string encoding)

  Parses a base64 encoded string and returns a string of the decoded data.

- nothing parseBase64StringToString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- binary parseHexString (string hexstr)

  Parses a hex-encoded string and returns the binary object.

- nothing parseHexString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash parseURL (string url, bool keep_brackets=False)

  Parses a URL string and returns a hash of the components; if the URL cannot be parsed then NOTHING is returned.

- nothing parseURL ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- hash parse_url (string url, bool keep_brackets=False)

  Parses a URL string and returns a hash of the components; throws an exception if the string cannot be parsed as a URL.

- string splice (string str)

  This function always returns an empty string "".

- string splice (string str, softint start)

  Returns a string based on the argument string but with characters removed from a certain character index.

- string splice (string str, softint start, softint len, __7_ string nstr)

  Returns a string based on the argument string but optionally with characters removed and/or added from a certain character index.

- list splice (list l, softint start)

  Returns a list based on the argument list but with elements removed from the given index to the end of the list.

- list splice (list l, softint start, softint len, __7_ softlist nlist)

  Returns a list based on the argument list but optionally with elements removed and/or added from a certain index.

- nothing splice ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int strtoint (string num, softint base=10)

  parses a string representing a number in a configurable base and returns the integer

- nothing strtoint ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- any callObjectMethod (object obj, string method,...)

  Calls a method of an object, passing the remainder of the arguments to the function as arguments to the method.

- nothing callObjectMethod ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- any callObjectMethodArgs (object obj, string method, __7_ softlist varg)

  Calls a method of an object, using the optional third argument as the argument list to the method.

- nothing callObjectMethodArgs ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- any call_pseudo (any val, string meth,...)

  calls a pseudo-method on the given value

- any call_pseudo_args (any val, string meth, __7_ softlist argv)

  calls a pseudo-method on the given value with arguments given as a list

- list getMethodList (object obj)

  Returns a list of strings of the names of the methods of the class of the object passed as a parameter.

- nothing getMethodList ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash getgrgid (softint gid)

  Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist NOTHING is returned.

- hash getgrgid2 (softint gid)

  Returns a group information hash representing the group information for the group ID passed, or, if the group ID does not exist, a `GETGRGID2-ERROR` exception is thrown.

- __7_ hash getgrnam (string name)

  Returns a group information hash representing the group information for the group name passed, or, if the group does not exist NOTHING is returned.

- hash getgrnam2 (string name)

  Returns a group information hash representing the group information for the group name passed, or, if the group does not exist, a `GETGRNAM2-ERROR` exception is thrown.

- hash getpwnam (string name)

  Returns a password information hash representing the user information for the user name passed, or, if the user does not exist NOTHING is returned.

- hash getpwnam2 (string name)

  Returns a password information hash representing the user information for the user name passed, or, if the user does not exist, a `GETPWNAM2-ERROR` exception is thrown.

- nothing getpwuid ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ hash getpwuid (softint uid)

  Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist NOTHING is returned.

- hash getpwuid2 (softint uid)

  Returns a password information hash representing the user information for the user ID passed, or, if the user ID does not exist, a `GETPWUID2-ERROR` exception is thrown.

- int bindex (softstring str, softstring substr, softint pos=0)

  Retrieves the byte position of a substring within a string.

- int bindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int brindex (softstring str, softstring substr, softint pos=-1)

  Retrieves the byte position of a substring within a string, starting the search from the end of the string.

- int brindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string chomp (string str)

Removes the trailing end-of-line indicator ("\n" or "\r\n") from a string and returns the new string (also see the chomp operator)

- __7_ string chomp (reference str)

  Removes the trailing end-of-line indicator ("\n" or "\r\n") from a reference to a string and returns the new string (also see the chomp operator)

- nothing chomp ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string chr (softint val, __7_ string encoding)

  Returns a string containing a single ASCII character represented by the numeric value passed.

- string chr (any arg)

  This function variant returns a string with a single ASCII NULL ('\0'); it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- nothing chr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string convert_encoding (string str, string encoding)

  Performs explicit string character encoding conversions.

- nothing convert_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string f_printf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string; enforces field widths on arguments larger than the given field width.

- string f_printf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string f_sprintf (string fmt,...)

  Returns a formatted string based on a format string and other arguments; enforces field widths on arguments larger than the given field width.

- string f_sprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string f_vprintf (string fmt, any varg)

  Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; enforces field widths on arguments larger than the given field width.

- string f_vsprintf (string fmt, any varg)

  Returns a formatted string based on a format string and other arguments given as a list after the format string; enforces field widths on arguments larger than the given field width.

- nothing flush ()

  Flushes output to the console output with print(), printf(), etc.

- string force_encoding (string str, string encoding)

  Returns the first string argument tagged with the character encoding given as the second argument; does not actually change the string data.

- nothing force_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string format_number (string fmt, softfloat num)

  Returns a string of a formatted number according to a number argument and a format string.

- nothing format_number ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string get_encoding (string str)

Returns a string describing the character encoding of the string argument passed.

- nothing get_encoding ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int index (softstring str, softstring substr, softint pos=0)

  Retrieves the character position of a substring within a string.

- int index ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string join (string str,...)

  Creates a string from separator string and a list of arguments.

- string join (string str, list l)

  Creates a string from separator string and a list of arguments.

- nothing join ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int length (softstring str)

  Returns the length in characters for the string passed.

- int length (binary bin)

  Returns the number of bytes in the binary object passed as an argument.

- nothing length ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int length (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int ord (softstring str, softint offset=0)

  Gives the numeric value of the given byte in the string passed; if no string is passed or the offset is after the end of the string, -1 is returned.

- int ord ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool parse_boolean (string str)

  tries to parse a string value as a boolean

- bool parse_boolean (...)

  returns the first value passed as a boolean

- nothing print (...)

  Outputs a string to standard output with no formatting.

- string printf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string; does not enforce field widths on arguments larger than the given field width.

- string printf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool regex (string str, string regex, int options=0)

  Returns True if the regular expression matches the string passed, otherwise returns False.

- nothing regex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ list regex_extract (string str, string regex, int options=0)

  Returns a list of substrings in a string based on matching patterns defined by a regular expression.

- nothing regex_extract ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string regex_subst (string str, string regex, string subst, int options=0)

  Returns a string with patterns substituted according to the arguments passed.

- nothing regex_subst ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string replace (string str, string source, string target, int start=0, int end=-1)

  Replaces all occurrences of a substring in a string with another string.

- nothing replace ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string reverse (softstring str)

  Reverses a string and returns the new string.

- int rindex (softstring str, softstring substr, softint pos=-1)

  Retrieves the character position of a substring within a string, starting the search from the end of the string.

- int rindex ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- list split (string sep, string str, bool with_separator=False)

  Splits a string into a list of components based on a separator string.

- list split (string sep, string str, string quote, bool trim_unquoted=False)

  Splits a string into a list of components based on a separator string and a quote character.

- list split (binary sep, binary data)

  Returns a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed.

- list split ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string sprintf (string fmt,...)

  Returns a formatted string based on a format string and other arguments; does not enforce field widths on arguments larger than the given field width.

- string sprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int strlen (softstring str)

  Returns the length in bytes of the string argument.

- nothing strlen ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int strlen (any arg)

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string strmul (softstring str, softint smul, __7_ softint offset)

  Returns a new string with a repeated string element and optionally removing trailing characters.

- string substr (softstring str, softint start)

  Returns a portion of a string starting from an integer offset.

- string substr (softstring str, softint start, softint len)

  Returns a portion of a string starting from an integer offset, with a length parameter.

- binary substr (binary b, softint start)

  Returns a portion of a binary object starting from an integer offset.

- binary substr (binary b, softint start, softint len)

  Returns a portion of a binary object starting from an integer offset, with a length parameter.

- nothing substr ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string tolower (string str)

  Returns a string in all lower-case characters based on the argument passed.
- nothing tolower ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string toupper (string str)

  Returns a string in all upper-case characters based on the argument passed.
- nothing toupper ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string trim (string str, __7_ string chars)

  Removes byte characters from the start and end of a string and returns the new string (also see the trim operator)
- __7_ string trim (reference str, __7_ string chars)

  Removes byte characters from the start and end of a reference to an lvalue containing a string and returns string after processing (also see the trim operator)
- nothing trim ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string trunc_str (softstring str, softint len, __7_ string encoding)

  Returns a truncated string with no more than the given number of bytes and optionally converted to a specific character encoding.
- string vprintf (string fmt,...)

  Outputs the string passed to standard output, using the first argument as a format string and a second argument giving a list or a single argument to the format string; does not enforce field widths on arguments larger than the given field width.
- string vprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string vsprintf (string fmt, any varg)

  Returns a formatted string based on a format string and other arguments given as a list after the format string; does not enforce field widths on arguments larger than the given field width.
- string vsprintf ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing delete_all_thread_data ()

  Deletes all keys in the thread-local data hash.
- nothing delete_thread_data (...)

  Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.
- nothing delete_thread_data (list l)

  Deletes the data associated to one or more keys in the thread-local data hash; if the data is an object, then it is destroyed.
- hash getAllThreadCallStacks ()

  Returns a hash of call stacks keyed by each TID (thread ID)
- hash get_all_thread_data ()

  Returns the entire thread-local data hash.
- any get_thread_data (string key)

  Returns the value of the thread-local data attached to the key passed.
- nothing get_thread_data ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- __7_ TimeZone get_thread_tz ()

  Returns any TimeZone set for the current thread, NOTHING if none is set.
- int gettid ()

  Returns the Qore thread ID (TID) of the current thread.
- nothing mark_thread_resources ()

  Marks thread resources so that any thread resources left allocated after this call will be cleaned up when throw_thread_resource_exceptions_to_mark() is called.
- int num_threads ()

  Returns the current number of threads in the process (not including the special signal handling thread)
- __7_ hash remove_thread_data (...)

  Removes the data associated to one or more keys in the thread-local data hash and returns the data removed.
- hash remove_thread_data (list l)

  Removes the data associated to one or more keys in the thread-local data hash from a literal list passed as the first argument and returns the data removed.
- nothing save_thread_data (hash h)

  Saves the data passed in the thread-local hash; all keys are merged into the thread-local hash, overwriting any information that may have been there before.
- nothing save_thread_data (string key, any value)

  Saves the data passed against the key passed as an argument in thread-local storage.
- nothing save_thread_data ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- bool set_thread_init (code init)

  Sets a call reference or closure to run every time a new thread is started.
- nothing set_thread_tz (TimeZone zone)

  Sets the default time zone for the current thread.
- nothing set_thread_tz ()

  Clears the thread-local time zone for the current thread; after this call TimeZone::get() will return the value set for the current Program.
- list thread_list ()

  Returns a list of all current thread IDs.
- nothing throwThreadResourceExceptions ()

  Immediately runs all thread resource cleanup routines for the current thread and throws all associated exceptions.
- bool throw_thread_resource_exceptions_to_mark ()

  Immediately runs all thread resource cleanup routines for the current thread for thread resources created since the last call to mark_thread_resources() and throws all associated exceptions.
- int clock_getmicros ()

  Returns an integer representing the system time in microseconds (1/1000000 second intervals) since Jan 1, 1970 00:00:00Z.
- int clock_getmillis ()

  Returns an integer representing the system time in milliseconds (1/1000 second intervals since Jan 1, 1970 00:00)
- int clock_getnanos ()

  Returns an integer representing the system time in nanoseconds (1/1000000000 second intervals) since Jan 1, 1970 00:00:00Z.
- date date (date dt)

  Returns the date passed.
- date date (string dtstr)

  Converts the argument to a date and returns the date.
- date date (float f)

  The argument is assumed to be the number of seconds and fractions of a second since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.
- date date (softint i)

The argument is assumed to be the number of seconds since 1970-01-01 in the local time zone; this value is used to produce the date value that is returned.

- date date ()

  This function just returns 1970-01-01Z.

- date date (null null)

  This function just returns 1970-01-01Z.

- date date (string dtstr, string mask)

  Returns the date/time value corresponding to parsing a string argument according to a format mask.

- hash date_info (date dt)

  Returns a hash of broken-down date/time information for the given date argument (can be either a relative or absolute date)

- hash date_info ()

  Returns a hash of broken-down date/time information for the current date and time.

- date date_ms (softint ms)

  Converts an integer argument representing the offset in milliseconds from January 1, 1970 in the local time zone to a date in the local time zone.

- nothing date_ms ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- date date_us (softint us)

  Converts an integer argument representing the offset in microseconds from January 1, 1970 in the local time zone to a date in the local time zone.

- date days (softint days)

  Returns a relative date/time value in days based on the integer argument passed to be used in date arithmetic.

- nothing days ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string format_date (string format, date dt)

  Returns a formatted string for a date argument passed.

- nothing format_date ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- date getDateFromISOWeek (softint year, softint week, softint day=1)

  Retuns an absolute date value for the `ISO-8601 calendar week information` passed (year, week number, optional: weekday, where 1=Monday, 7=Sunday) in the current time zone.

- int getDayNumber (date dt)

  Returns an integer representing the ordinal day number in the year (corresponding to the `ISO-8601 day number`) for the absolute date value passed.

- nothing getDayNumber ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int getDayOfWeek (date dt)

  Returns an integer representing the day of the week for the absolute date value passed (0=Sunday, 6=Saturday)

- nothing getDayOfWeek ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int getISODayOfWeek (date dt)

  Returns an integer representing the ISO-8601 day of the week for the absolute date value passed (1=Monday, 7=Sunday)

- nothing getISODayOfWeek ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- hash getISOWeekHash (date dt)

Returns a hash representing the ISO-8601 calendar week information for the absolute date passed (hash keys: "year", "week", "day")

- nothing getISOWeekHash ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- string getISOWeekString (date dt)

  Returns a string representing the ISO-8601 calendar week information for the absolute date passed (ex: 2006-01-01 = "2005-W52-7")

- nothing getISOWeekString ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_days (date dt)

  Returns an integer corresponding to the literal day value in the date (does not calculate a duration)

- nothing get_days ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_duration_microseconds (date dt)

  Returns an integer value representing the the number of microseconds of duration in the value of the date passed (can be either a relative or absolute date)

- int get_duration_milliseconds (date dt)

  Returns an integer value representing the the number of milliseconds of duration in the value of the date passed (can be either a relative or absolute date)

- int get_duration_seconds (date dt)

  Returns an integer value representing the the number of seconds of duration in the value of the date passed (can be either a relative or absolute date)

- int get_epoch_seconds (date dt)

  Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.

- nothing get_epoch_seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_hours (date dt)

  Returns an integer corresponding to the literal hour value in the date (does not calculate a duration)

- nothing get_hours ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_microseconds (date dt)

  Returns an integer corresponding to the literal microsecond value in the date (does not calculate a duration)

- date get_midnight (date dt)

  Returns midnight on the date passed (strips the time component on the new value)

- nothing get_midnight ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_milliseconds (date dt)

  Returns an integer corresponding to the literal millisecond value in the date (does not calculate a duration)

- nothing get_milliseconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_minutes (date dt)

  Returns an integer corresponding to the literal minute value in the date (does not calculate a duration)

- nothing get_minutes ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_months (date dt)

Returns an integer corresponding to the literal month value in the date (does not calculate a duration)

- nothing get_months ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_seconds (date dt)

  Returns an integer corresponding to the literal second value in the date (does not calculate a duration)

- nothing get_seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int get_years (date dt)

  Returns an integer corresponding to the literal year value in the date (does not calculate a duration)

- nothing get_years ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- date gmtime ()

  Returns the current UTC (GMT) time with a resolution of a second.

- date gmtime (softint secs, softint us=0)

  Returns a date/time value in UTC (GMT) from arguments giving the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)

- date gmtime (date dt)

  Returns the date and time in UTC (GMT) corresponding to the date argument passed.

- date hours (softint hours)

  Returns a relative date/time value in hours based on the integer argument passed to be used in date arithmetic.

- nothing hours ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool is_date_absolute (date dt)

  Returns True if the argument is an absolute date/time value, False if not.

- bool is_date_absolute ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- bool is_date_relative (date dt)

  Returns True if the argument is an relative date/time value, False if not.

- bool is_date_relative ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- date localtime ()

  Returns the current date and time with a resolution to the second.

- date localtime (softint secs, softint us=0)

  Returns the date and time in the local time zone corresponding to the integer arguments passed, which are interpreted as the number of seconds and microseconds since Jan 1, 1970, 00:00:00 Z (UTC)

- date localtime (date dt)

  Returns the date and time in the local time zone corresponding to the date argument passed.

- date microseconds (softint us)

  Returns a relative date/time value in microseconds based on the integer argument passed to be used in date arithmetic.

- date milliseconds (softint ms)

  Returns a relative date/time value in milliseconds based on the integer argument passed to be used in date arithmetic.

- nothing milliseconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- date minutes (softint minutes)

  Returns a relative date/time value in minutes based on the integer argument passed to be used in date arithmetic.

- nothing minutes ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int mktime (date dt)

  Returns the number of seconds of the date and time in local time passed since Jan 1, 1970, 00:00:00 Z (UTC); negative values are returned for dates before the epoch.
- nothing mktime ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date months (softint months)

  Returns a relative date/time value in months based on the integer argument passed to be used in date arithmetic.
- nothing months ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date now ()

  Returns the current date and time with a resolution to the second.
- date now_ms ()

  Returns the current date and time with a resolution to the millisecond.
- date now_us ()

  Returns the current date and time with a resolution to the microsecond.
- date now_utc ()

  Returns the current UTC date and time with a resolution to the microsecond.
- date seconds (softint seconds)

  Returns a relative date/time value in seconds based on the integer argument passed to be used in date arithmetic.
- nothing seconds ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int timegm (date dt)

  Returns the number of seconds since January 1, 1970 00:00:00 in the local time zone for the given date.
- nothing timegm ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- date years (softint years)

  Returns a relative date/time value in years based on the integer argument passed to be used in date arithmetic.
- nothing years ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- binary binary ()

  Always returns an empty binary object (of zero length)
- binary binary (null x)

  Always returns an empty binary object (of zero length)
- binary binary (softstring str)

  Returns a binary data type of the string passed; data types other than string will first be converted to a string and then returned as binary data.
- binary binary (binary bin)

  Always returns the same binary object passed.
- string binary_to_string (binary b, __7_ string encoding)

  Returns a string created from the binary data passed, taking an optional second argument giving the string encoding; if no second argument is passed then the default character encoding is assumed.
- bool boolean (any arg)

  Converts the argument to a boolean value.
- float float (softfloat f)

Converts the argument to a floating-point (float) value.

- float float ()

  Always returns 0.0.

- hash hash (object obj)

  Returns a hash of an object's members.

- hash hash (list l)

  Returns a hash by taking even numbered list elements (starting with 0) and converting them to strings for the hash keys, and the odd numbered elements following the keys as the key value.

- hash hash (list keys, list values)

  Returns a hash by taking the first list as a list of keys, and the second list as a list of values.

- hash hash (hash h)

  Returns itself.

- hash hash ()

  Always returns the same hash passed.

- int int (string str, softint base)

  Converts the argument to an integer value.

- int int (softint i)

  Converts the argument to an integer value.

- int int ()

  Always returns 0.

- list list (...)

  Returns a list of the arguments passed at the top level.

- number number (softnumber n)

  Converts the argument to a number value.

- number number ()

  Always returns 0.0.

- string string (softstring str)

  Converts the argument to a string.

- string string ()

  Always returns an empty string.

- string type (any arg)

  Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

- string typename (any arg)

  Returns a string giving the data type of the argument passed; see String Type Constants for the values returned by this function.

Variables

- const NF_Default = QORE_NF_DEFAULT

  for the default format with a rounding heuristic to try to remove noise in insignificant digits from the binary to digital conversion of high-precision numbers

- const NF_Raw = QORE_NF_RAW

  for the raw format without the noise reduction heuristic in the NF_Default format

- const NF_Scientific = QORE_NF_SCIENTIFIC

  for the scientific format (exponential notation)

- const O_ACCMODE = O_ACCMODE

  Mask for access modes (O_RDONLY|O_WRONLY|O_RDWR)

- const O_APPEND = O_APPEND

  Open the file in append mode (append on each write)

- const O_CREAT = O_CREAT

Create the file if it doesn't exist.

- const O_DIRECT = O_DIRECT

  direct disk access hint (0 on platforms where this is not available)
- const O_DIRECTORY = O_DIRECTORY

  must be a directory (0 on platforms where this is not available)
- const O_EXCL = O_EXCL

  Raise an error if used with O_CREAT and the file exists.
- const O_NDELAY = O_NDELAY

  synonym for O_NONBLOCK (untested with Qore; 0 on platforms where this is not available)
- const O_NOCTTY = O_NOCTTY

  don't allocate controlling tty (0 on platforms where this is not available)
- const O_NOFOLLOW = O_NOFOLLOW

  don't follow links (0 on platforms where this is not available)
- const O_NONBLOCK = O_NONBLOCK

  non-blocking I/O (untested with Qore; 0 on platforms where this is not available)
- const O_RDONLY = O_RDONLY

  Open the file read-only.
- const O_RDWR = O_RDWR

  Open for reading and writing.
- const O_SYNC = O_SYNC

  synchronized file update option (0 on platforms where this is not available)
- const O_TRUNC = O_TRUNC

  Truncate the size to zero.
- const O_WRONLY = O_WRONLY

  Open the file write-only.
- const F_RDLCK = F_RDLCK

  Use for read-only locking.
- const F_UNLCK = F_UNLCK

  Use for unlocking a lock.
- const F_WRLCK = F_WRLCK

  Use for exclusive write locking.
- const SEEK_CUR = SEEK_CUR

  Indicates that the offset is from the current position in the file.
- const SEEK_END = SEEK_END

  Indicates that the offset is from the end of the file.
- const SEEK_SET = SEEK_SET

  Indicates that the offset is from the start of the file.
- const PO_ALLOW_BARE_REFS = PO_ALLOW_BARE_REFS

  Prohibits the use of the '$' character in variable names, method calls, and object member references.
- const PO_ASSUME_LOCAL = PO_ASSUME_LOCAL

  Assume local variable scope when variables are first referenced if no ***my*** `or` ***our*** `is` *present.*
- const PO_DEFAULT = PO_DEFAULT

  This option is the empty option, meaning no options are set.
- const PO_FREE_OPTIONS = PO_FREE_OPTIONS

  mask of options that have no effect on code access or code safety but just affect programming style
- const PO_IN_MODULE = PO_IN_MODULE

  Only set by the system when in a user module Program.
- const PO_LOCKDOWN = PO_LOCKDOWN

  Sets very restrictive access; this restriction is designed to allow code to only execute logic, no I/O, no threading, no external access.
- const PO_LOCK_WARNINGS = PO_LOCK_WARNINGS

Disallows changes to the warning mask.

- const PO_NEW_STYLE = PO_NEW_STYLE

  Set a more C++ or Java type programming style; prohibits usage of the "$" character and also assumes local variable scope without ***my.***

- const PO_NO_CHILD_PO_RESTRICTIONS = PO_NO_CHILD_PO_RESTRICTIONS

  Allows child program objects to have fewer parse restrictions (i.e. more capabilies) than the parent object.

- const PO_NO_CLASS_DEFS = PO_NO_CLASS_DEFS

  Disallows class definitions.

- const PO_NO_CONSTANT_DEFS = PO_NO_CONSTANT_DEFS

  Disallows constant definitions.

- const PO_NO_DATABASE = PO_NO_DATABASE

  Disallows access to database functionality.

- const PO_NO_EMBEDDED_LOGIC = PO_NO_EMBEDDED_LOGIC

  Prohibits embedded logic from being used.

- const PO_NO_EXTERNAL_ACCESS = PO_NO_EXTERNAL_ACCESS

  Prohibits any external access.

- const PO_NO_EXTERNAL_INFO = PO_NO_EXTERNAL_INFO

  Disallows access to functionality that provides information about the computing environment.

- const PO_NO_EXTERNAL_PROCESS = PO_NO_EXTERNAL_PROCESS

  Disallows any access to external processes (with system(), backquote(), exec(), etc)

- const PO_NO_FILESYSTEM = PO_NO_FILESYSTEM

  Disallows access to the filesystem.

- const PO_NO_GLOBAL_VARS = PO_NO_GLOBAL_VARS

  Disallows the use of global variables.

- const PO_NO_GUI = PO_NO_GUI

  Disallows access to functionality that draws graphics to the display.

- const PO_NO_INHERIT_GLOBAL_VARS = PO_NO_INHERIT_GLOBAL_VARS

  Precludes global variables from being inherited into the new Program object.

- const PO_NO_INHERIT_USER_FUNC_VARIANTS = PO_NO_INHERIT_USER_FUNC_VARIANTS

  Precludes public user function variants from being inherited into the new Program object.

- const PO_NO_IO = PO_NO_IO

  Prohibits all terminal and file I/O and GUI operations.

- const PO_NO_LOCALE_CONTROL = PO_NO_LOCALE_CONTROL

  Disallows access to functionality that can change locale parameters.

- const PO_NO_MODULES = PO_NO_MODULES

  Disallows loading modules with the %requires directive or at runtime with load_module()

- const PO_NO_NAMESPACE_DEFS = PO_NO_NAMESPACE_DEFS

  Disallows new namespace definitions.

- const PO_NO_NETWORK = PO_NO_NETWORK

  Disallows access to network functionality.

- const PO_NO_NEW = PO_NO_NEW

  Disallows use of the new operator.

- const PO_NO_PROCESS_CONTROL = PO_NO_PROCESS_CONTROL

  Disallows access to functions that would affect the current process (exit(), exec(), fork(), etc)

- const PO_NO_SUBROUTINE_DEFS = PO_NO_SUBROUTINE_DEFS

  Disallows subroutine (function) definitions.

- const PO_NO_SYSTEM_CLASSES = PO_NO_SYSTEM_CLASSES

  Prohibits system classes from being imported into the new Program object.

- const PO_NO_SYSTEM_FUNC_VARIANTS = PO_NO_SYSTEM_FUNC_VARIANTS

  Prohibits builtin/system function variants from being imported into the new Program object.

- const PO_NO_TERMINAL_IO = PO_NO_TERMINAL_IO

Disallows access to reading from and/or writing to the terminal.

- const PO_NO_THREADS = PO_NO_THREADS

  Prohibits access to all threading information.

- const PO_NO_THREAD_CLASSES = PO_NO_THREAD_CLASSES

  Disallows access to any thread classes.

- const PO_NO_THREAD_CONTROL = PO_NO_THREAD_CONTROL

  Disallows access to any thread-control functions and thread-relevant statements and operators (for example the background operator and the thread_exit statement)

- const PO_NO_THREAD_INFO = PO_NO_THREAD_INFO

  Disallows access to functionality that provides information about threading.

- const PO_NO_TOP_LEVEL_STATEMENTS = PO_NO_TOP_LEVEL_STATEMENTS

  Disallows top level code.

- const PO_NO_USER_CLASSES = PO_NO_USER_CLASSES

  Prohibits user classes from being imported into the new Program object.

- const PO_POSITIVE_OPTIONS = PO_POSITIVE_OPTIONS

  mask of all parse options allowing for more freedom (instead of less)

- const PO_REQUIRE_OUR = PO_REQUIRE_OUR

  Requires global variables to be declared with our before use.

- const PO_REQUIRE_PROTOTYPES = PO_REQUIRE_PROTOTYPES

  Requires all function and method parameters and return types to have type declarations.

- const PO_REQUIRE_TYPES = PO_REQUIRE_TYPES

  Requires all function and method parameters, return types, variables, and object members to have type declarations.

- const PO_STRICT_ARGS = PO_STRICT_ARGS

  Prohibits access to builtin functions and methods flagged with RT_NOOP and also causes errors to be raised if excess arguments are given to functions that do not access excess arguments.

- const PO_STRICT_BOOLEAN_EVAL = PO_STRICT_BOOLEAN_EVAL

  Sets strict mathematical boolean evaluation runtime mode (the qore default prior to v0.8.6)

- const WARN_ALL = QP_WARN_ALL

  Enables all warnings.

- const WARN_CALL_WITH_TYPE_ERRORS = QP_WARN_CALL_WITH_TYPE_ERRORS

  Enables warnings when the parser determines that the argument types of a function or method call are such that the operation is guaranteed to produce a constant value.

- const WARN_DEFAULT = QP_WARN_DEFAULT

  The default warning mask.

- const WARN_DEPRECATED = QP_WARN_DEPRECATED

  Enables a warning when deprecated code is used.

- const WARN_DUPLICATE_BLOCK_VARS = QP_WARN_DUPLICATE_BLOCK_VARS

  Enables a warning when a program declares a local variable more than once in the same block; note that this is not a warning but rather an error when assume-local or new-style parse options are set.

- const WARN_DUPLICATE_GLOBAL_VARS = QP_WARN_DUPLICATE_GLOBAL_VARS

  Indicates that the embedded code has declared the same global variable more than once.

- const WARN_DUPLICATE_HASH_KEY = QP_WARN_DUPLICATE_HASH_KEY

  Enables a warning when an immediate hash is declared and at least one of the keys is repeated.

- const WARN_DUPLICATE_LOCAL_VARS = QP_WARN_DUPLICATE_LOCAL_VARS

  Enables a warning when a local variable with the same name is declared in a subblock (ie another local variable with the same name is reachable in the same lexical scope); note that this warning can raise false positives if the programmer is used to redeclaring the same variable names in subblocks.

- const WARN_EXCESS_ARGS = QP_WARN_EXCESS_ARGS

  Enables a warning when a function or method call is made with more arguments than are used by the function or method.

- const WARN_INVALID_OPERATION = QP_WARN_INVALID_OPERATION

Indicates that the embedded code performs some operation that is guaranteed to produce no result (for example, using the [] operator on an integer value)

- const WARN_MODULES = QP_WARN_MODULES

  The default warning mask for user modules.
- const WARN_NONE = QP_WARN_NONE

  Represents no warning.
- const WARN_NONEXISTENT_METHOD_CALL = QP_WARN_NONEXISTENT_METHOD_CALL

  Indicates that the embedded code is calling an unknown method in a class.
- const WARN_RETURN_VALUE_IGNORED = QP_WARN_RETURN_VALUE_IGNORED

  Enables a warning when a function or method call is made with no side effects and the return value is ignored.
- const WARN_UNDECLARED_VAR = QP_WARN_UNDECLARED_VAR

  Indicates that the embedded code referenced an undeclared variable that will be assumed to be a global variable.
- const WARN_UNKNOWN_WARNING = QP_WARN_UNKNOWN_WARNING

  Indicates that the embedded code tried to enable or disable an unknown warning.
- const WARN_UNREACHABLE_CODE = QP_WARN_UNREACHABLE_CODE

  Indicates that code cannot be reached (for example; code in the same local block after an unconditional return or thread_exit statement)
- const WARN_UNREFERENCED_VARIABLE = QP_WARN_UNREFERENCED_VARIABLE

  This warning is raised when a variable is declared in a block but never referenced.
- const WARN_WARNING_MASK_UNCHANGED = QP_WARN_WARNING_MASK_UNCHANGED

  This warning means that the embedded code tried to change the warning mask, but it was locked, so the warning mask was actually unchanged.
- const NT_BINARY = NT_BINARY

  type code for binary values
- const NT_BOOLEAN = NT_BOOLEAN

  type code for boolean values
- const NT_CALLREF = NT_FUNCREF

  type code for call references
- const NT_CLOSURE = NT_RUNTIME_CLOSURE

  type code for closures
- const NT_DATE = NT_DATE

  type code for date/time values
- const NT_FLOAT = NT_FLOAT

  type code for float values
- const NT_HASH = NT_HASH

  type code for hash values
- const NT_INT = NT_INT

  type code for integer values
- const NT_LIST = NT_LIST

  type code for list values
- const NT_NOTHING = NT_NOTHING

  type code for no value (NOTHING)
- const NT_NULL = NT_NULL

  type code for NULL
- const NT_NUMBER = NT_NUMBER

  type code for number values
- const NT_OBJECT = NT_OBJECT

  type code for objects
- const NT_STRING = NT_STRING

  type code for string values
- const TypeCodeMap

type code map, looks up type names from type code values

- const TypeNameMap

  type name map, looks up type codes from type names
- const False = bool(false)

  logical False
- const True = bool(true)

  logical True
- const NOTHING = qore(&Nothing)

  a constant representing the lack of a value
- const NULL = qore(&Null)

  logical False
- const ET_System = "System"

  Exception type code system exceptions (thrown in internal Qore code or in modules)
- const ET_User = "User"

  Exception type for user exceptions (thrown by the throw statement)
- const CT_Builtin = CT_BUILTIN

  Call type for builtin code.
- const CT_NewThread = CT_NEWTHREAD

  Call type for the start of a new thread by the background operator.
- const CT_Rethrow = CT_RETHROW

  Call type for an exception thrown by the rethrow statement.
- const CT_User = CT_USER

  Call type for user code.
- const Build = qore(new QoreBigIntNode(qore_build_number))

  The integer Qore build number.
- const BuildHost = qore(new QoreStringNode(qore_build_host))

  The host name of the host used to build the Qore library.
- const CFLAGS = qore(new QoreStringNode(qore_cflags))

  A string giving the C++ compiler flags used to build Qore.
- const Compiler = qore(new QoreStringNode(qore_cplusplus_compiler))

  A string giving the C++ compiler used to build Qore.
- const LDFLAGS = qore(new QoreStringNode(qore_ldflags))

  A string giving the linker flags used to build Qore.
- const MACHINE_MSB = bool(Q_MACHINE_MSB)

  True if the current machine uses `big-endian or MSB byte order` *or False if the current machine uses* `little-endian or LSB byte order`
- const PlatformCPU = qore(new QoreStringNode(TARGET_ARCH))

  The string for the platform's CPU architecture.
- const PlatformOS = qore(new QoreStringNode(TARGET_OS))

  A string giving the platform operating-system name.
- const VersionMajor = qore(new QoreBigIntNode(qore_version_major))

  The integer Qore major version number.
- const VersionMinor = qore(new QoreBigIntNode(qore_version_minor))

  The integer Qore minor version number.
- const VersionString = qore(new QoreStringNode(qore_version_string))

  The full Qore version string.
- const VersionSub = qore(new QoreBigIntNode(qore_version_sub))

  The integer Qore sub version number.
- const SOURCE_FILE = QORE_SOURCE_FILE

  File class source code
- const SOURCE_FTPCLIENT = QORE_SOURCE_FTPCLIENT

FtpClient class source code

- const SOURCE_HTTPCLIENT = QORE_SOURCE_HTTPCLIENT

  HTTPClient class source code
- const SOURCE_SOCKET = QORE_SOURCE_SOCKET

  Socket class source code
- const EVENT_MAP

  Maps from Event Constants (the keys) to descriptive strings (the values)
- const EVENT_SOURCE_MAP

  Maps from Event Source Constants (the keys) to descriptive strings (the values)
- const EVENT_CHANNEL_CLOSED = QORE_EVENT_CHANNEL_CLOSED

  Raised when a socket or file is closed.
- const EVENT_CONNECTED = QORE_EVENT_CONNECTED

  Raised when the socket connection has been established.
- const EVENT_CONNECTING = QORE_EVENT_CONNECTING

  Raised right before a socket connection attempt is made.
- const EVENT_DATA_READ = QORE_EVENT_DATA_READ

  Raised when data has been read from a file.
- const EVENT_DATA_WRITTEN = QORE_EVENT_DATA_WRITTEN

  Raised when data has been written to a file.
- const EVENT_DELETED = QORE_EVENT_DELETED

  Raised when the object being monitored is deleted.
- const EVENT_FILE_OPENED = QORE_EVENT_FILE_OPENED

  Raised when a file has been successfully opened.
- const EVENT_FTP_MESSAGE_RECEIVED = QORE_EVENT_FTP_MESSAGE_RECEIVED

  Raised when an FTP reply is received on the control channel.
- const EVENT_FTP_SEND_MESSAGE = QORE_EVENT_FTP_SEND_MESSAGE

  Raised immediately before an FTP control message is sent.
- const EVENT_HOSTNAME_LOOKUP = QORE_EVENT_HOSTNAME_LOOKUP

  Raised when a hostname lookup is started.
- const EVENT_HOSTNAME_RESOLVED = QORE_EVENT_HOSTNAME_RESOLVED

  Raised when a hostname lookup is resolved.
- const EVENT_HTTP_CHUNKED_DATA_RECEIVED = QORE_EVENT_HTTP_CHUNKED_DATA_RECEIVED

  Raised when a block of HTTP chunked data is received.
- const EVENT_HTTP_CHUNKED_END = QORE_EVENT_HTTP_CHUNKED_END

  Raised when all HTTP chunked data has been received.
- const EVENT_HTTP_CHUNKED_START = QORE_EVENT_HTTP_CHUNKED_START

  Raised when HTTP chunked data is about to be received.
- const EVENT_HTTP_CHUNK_SIZE = QORE_EVENT_HTTP_CHUNK_SIZE

  Raised when the next chunk size for HTTP chunked data is known.
- const EVENT_HTTP_CONTENT_LENGTH = QORE_EVENT_HTTP_CONTENT_LENGTH

  Raised when the HTTP "`Content-Length`" header is received.
- const EVENT_HTTP_FOOTERS_RECEIVED = QORE_EVENT_HTTP_FOOTERS_RECEIVED

  Raised when HTTP footers are received.
- const EVENT_HTTP_MESSAGE_RECEIVED = QORE_EVENT_HTTP_MESSAGE_RECEIVED

  Raised when an HTTP message is received.
- const EVENT_HTTP_REDIRECT = QORE_EVENT_HTTP_REDIRECT

  Raised when an HTTP redirect message is received.
- const EVENT_HTTP_SEND_MESSAGE = QORE_EVENT_HTTP_SEND_MESSAGE

  Raised when an HTTP message is sent.
- const EVENT_OPEN_FILE = QORE_EVENT_OPEN_FILE

Raised right before a file is opened.

- const EVENT_PACKET_READ = QORE_EVENT_PACKET_READ

  Raised when a network packet is received.
- const EVENT_PACKET_SENT = QORE_EVENT_PACKET_SENT

  Raised when a network packet is sent.
- const EVENT_SSL_ESTABLISHED = QORE_EVENT_SSL_ESTABLISHED

  Raised when SSL communication has been negotiated and established.
- const EVENT_START_SSL = QORE_EVENT_START_SSL

  Raised when socket SSL negotiation starts.
- const stderr = qore(QC_FILE->execSystemConstructor(2))

  system constant for stderr (file descriptor 2)
- const stdin = qore(QC_FILE->execSystemConstructor(0))

  system constant for stdin (file descriptor 0)
- const stdout = qore(QC_FILE->execSystemConstructor(1))

  system constant for stdout (file descriptor 1)
- const S_IFBLK = S_IFBLK

  Bitmask signifying if the file is a block special (device) file.
- const S_IFCHR = S_IFCHR

  Bit signifying if the file is a character special (device) file.
- const S_IFDIR = S_IFDIR

  Bit signifying if the entry is a directory.
- const S_IFLNK = S_IFLNK

  Bitmask signifying if the file is a symbolic link; equal to 0 on native Windows ports.
- const S_IFMT = S_IFMT

  File type bitmask
- const S_IFREG = S_IFREG

  Bit signifying if the file is a regular file.
- const S_IFSOCK = S_IFSOCK

  Bitmask signifying if the file is a socket file; equal to 0 on native Windows ports.
- const S_IFWHT = S_IFWHT

  Bitmask signifying if the file is a whiteout file; equal to 0 on native Windows ports.
- const S_IRGRP = S_IRGRP

  Bit signifying if the file's group has read permissions; equal to 0 on native Windows ports.
- const S_IROTH = S_IROTH

  Bit signifying if other has read permissions; equal to 0 on native Windows ports.
- const S_IRUSR = S_IRUSR

  Bit signifying if the file's owner has read permissions.
- const S_IRWXG = S_IRWXG

  Bitmask giving the RWX mask for the group; equal to 0 on native Windows ports.
- const S_IRWXO = S_IRWXO

  Bitmask giving the RWX mask for other; equal to 0 on native Windows ports.
- const S_IRWXU = S_IRWXU

  Bitmask giving the RWX mask for the owner.
- const S_ISGID = S_ISGID

  Bit signifying set group id on execution; equal to 0 on native Windows ports.
- const S_ISUID = S_ISUID

  Bit signifying set user id on execution; equal to 0 on native Windows ports.
- const S_ISVTX = S_ISVTX

  Bit signifying restricted deletes for directories; equal to 0 on native Windows ports.
- const S_IWGRP = S_IWGRP

  Bit signifying if the file's group has write permissions; equal to 0 on native Windows ports.

- const S_IWOTH = S_IWOTH

  Bit signifying if other has write permissions; equal to 0 on native Windows ports.
- const S_IWUSR = S_IWUSR

  Bit signifying if the file's owner has write permissions.
- const S_IXGRP = S_IXGRP

  Bit signifying if the file's group has execute permissions; equal to 0 on native Windows ports.
- const S_IXOTH = S_IXOTH

  Bit signifying if other has execute permissions; equal to 0 on native Windows ports.
- const S_IXUSR = S_IXUSR

  Bit signifying if the file's owner has execute permissions.
- const X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH = "X509_V_ERR_AKID_ISSUER_SERIAL_MI↩SMATCH"

  Issuer name and serial number of candidate certificate do not match the authority key identifier of the current certificate.
- const X509_V_ERR_AKID_SKID_MISMATCH = "X509_V_ERR_AKID_SKID_MISMATCH"

  The current candidate issuer certificate was rejected because its subject key identifier was present and did not match the authority key identifier of the current certificate.
- const X509_V_ERR_APPLICATION_VERIFICATION = "X509_V_ERR_APPLICATION_VERIFICATION"

  Verification failure.
- const X509_V_ERR_CERT_CHAIN_TOO_LONG = "X509_V_ERR_CERT_CHAIN_TOO_LONG"

  Certificate chain too long.
- const X509_V_ERR_CERT_HAS_EXPIRED = "X509_V_ERR_CERT_HAS_EXPIRED"

  Certificate has expired.
- const X509_V_ERR_CERT_NOT_YET_VALID = "X509_V_ERR_CERT_NOT_YET_VALID"

  Certificate is not yet valid.
- const X509_V_ERR_CERT_REJECTED = "X509_V_ERR_CERT_REJECTED"

  Root CA is marked to reject the specified purpose.
- const X509_V_ERR_CERT_REVOKED = "X509_V_ERR_CERT_REVOKED"

  Certificate has been revoked.
- const X509_V_ERR_CERT_SIGNATURE_FAILURE = "X509_V_ERR_CERT_SIGNATURE_FAILURE"

  Certificate signature failure; the signature of the certificate is invalid.
- const X509_V_ERR_CERT_UNTRUSTED = "X509_V_ERR_CERT_UNTRUSTED"

  Root CA is not marked as trusted for the specified purpose.
- const X509_V_ERR_CRL_HAS_EXPIRED = "X509_V_ERR_CRL_HAS_EXPIRED"

  CRL has expired.
- const X509_V_ERR_CRL_NOT_YET_VALID = "X509_V_ERR_CRL_NOT_YET_VALID"

  CRL is not yet valid.
- const X509_V_ERR_CRL_SIGNATURE_FAILURE = "X509_V_ERR_CRL_SIGNATURE_FAILURE"

  CRL signature failure; the signature of the certificate is invalid.
- const X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT = "X509_V_ERR_DEPTH_ZERO_SELF_SI↩GNED_CERT"

  Certificate is self-signed and cannot be found in the trusted list.
- const X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD = "X509_V_ERR_ERROR_IN_CERT_NO↩T_AFTER_FIELD"

  Format error in certificate's notAfter field (invalid time)
- const X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD = "X509_V_ERR_ERROR_IN_CERT_N↩OT_BEFORE_FIELD"

  Format error in certificate's notBefore field (invalid time)
- const X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD = "X509_V_ERR_ERROR_IN_CRL_LAS↩T_UPDATE_FIELD"

  Format error in CRL's lastUpdate field (invalid time)

- const X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD = "X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD"

  Format error in CRL's nextUpdate field (invalid time)
- const X509_V_ERR_INVALID_CA = "X509_V_ERR_INVALID_CA"

  Invalid CA certificate.
- const X509_V_ERR_INVALID_PURPOSE = "X509_V_ERR_INVALID_PURPOSE"

  The certificate cannot be used for the specified purpose.
- const X509_V_ERR_KEYUSAGE_NO_CERTSIGN = "X509_V_ERR_KEYUSAGE_NO_CERTSIGN"

  The keyUsage extension does not permit certificate signing.
- const X509_V_ERR_OUT_OF_MEM = "X509_V_ERR_OUT_OF_MEM"

  Out of memory error.
- const X509_V_ERR_PATH_LENGTH_EXCEEDED = "X509_V_ERR_PATH_LENGTH_EXCEEDED"

  The basicConstraints pathlength parameter has been exceeded.
- const X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN = "X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN"

  Self signed certificate in certificate chain.
- const X509_V_ERR_SUBJECT_ISSUER_MISMATCH = "X509_V_ERR_SUBJECT_ISSUER_MISMATCH"

  The current candidate issuer certificate was rejected because its subject name did not match the issuer name of the current certificate.
- const X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY = "X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY"

  Unable to decode issuer public key (SubjectPublicKeyInfo)
- const X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE = "X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE"

  Unable to decrypt certificate's signature. This means that the actual signature value could not be determined rather than it not matching the expected value; this is only meaningful for RSA.
- const X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE = "X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE"

  Unable to decrypt CRL's signature.
- const X509_V_ERR_UNABLE_TO_GET_CRL = "X509_V_ERR_UNABLE_TO_GET_CRL"

  Unable to get certificate CRL.
- const X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT = "X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT"

  Unable to get issuer certificate.
- const X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY = "X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY"

  Unable to get local issuer certificate. This normally means the list of trusted certificates is not complete.
- const X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE = "X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE"

  Unable to verify the first certificate.
- const X509_V_OK = "X509_V_OK"

  Verification OK.
- const X509_VerificationReasons

  maps from varification strings to verification code descriptions
- const AFMap = qore(get_network_address_family_map())

  mapping from Network Address Family Constants to string codes
- const AFStrMap

  mapping from network address family string codes to Network Address Family Constants
- const AF_INET = AF_INET

  IPv4 address family.
- const AF_INET6 = AF_INET6

  IPv6 address family.

- const AF_LOCAL = AF_LOCAL

  POSIX synonym for AF_UNIX.
- const AF_UNIX = AF_UNIX

  UNIX domain address family (UNIX socket files)
- const AF_UNSPEC = AF_UNSPEC

  unspecified address family
- const AI_ADDRCONFIG = AI_ADDRCONFIG

  if this bit is set, addresses of each family are returned only if they are configured on the system
- const AI_ALL = AI_ALL

  If this bit is set along with AI_V4MAPPED then all matching IPv6 and IPv4 addresses are returned.
- const AI_CANONNAME = AI_CANONNAME

  If this bit is set, then getaddrinfo() will return the canonical name of the hostname in the "`canonname`" key of the first element returned.
- const AI_NUMERICHOST = AI_NUMERICHOST

  If this bit is set, then the host is assumed to be an address and no hostname lookup will be preformed.
- const AI_NUMERICSERV = AI_NUMERICSERV

  If this bit is set, then the service is assumed to be a numeric port string, and no service lookup will be performed.
- const AI_PASSIVE = AI_PASSIVE

  If this bit is set, then the returned information should be usable for a call to Socket::bind()
- const AI_V4MAPPED = AI_V4MAPPED

  If this bit is set, getaddrinfo() will return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses/.
- const IPPROTO_TCP = IPPROTO_TCP

  for the TCP protocol
- const IPPROTO_UDP = IPPROTO_UDP

  for the UDP protocol
- const SOCK_DGRAM = SOCK_DGRAM

  for datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length
- const SOCK_RAW = SOCK_RAW

  raw socket interface, only available to the superuser, untested
- const SOCK_STREAM = SOCK_STREAM

  for sequenced, reliable, two-way connection-based byte streams (the default)
- const ALTWERASE = ALTWERASE

  use alternate WERASE algorithm
- const ECHO = ECHO

  enable echoing
- const ECHOCTL = ECHOCTL

  echo control chars as ^(Char)
- const ECHOE = ECHOE

  visually erase chars
- const ECHOKE = ECHOKE

  visual erase for line kill
- const ECHONL = ECHONL

  echo NL even if ECHO is off
- const ECHOPRT = ECHOPRT

  visual erase mode for hardcopy
- const EXTPROC = EXTPROC

  external processing
- const FLUSHO = FLUSHO

  output being flushed (state)
- const ICANON = ICANON

  canonicalize input lines

- const IEXTEN = IEXTEN

  enable DISCARD and LNEXT
- const ISIG = ISIG

  enable signals INTR, QUIT, [D]SUSP
- const NOFLSH = NOFLSH

  don't flush after interrupt
- const NOKERNINFO = NOKERNINFO

  no kernel output from VSTATUS
- const PENDIN = PENDIN

  retype pending input (state)
- const TOSTOP = TOSTOP

  stop background jobs from output
- const CCAR_OFLOW = CCAR_OFLOW

  DCD flow control of output.
- const CCTS_OFLOW = CCTS_OFLOW

  CTS flow control of output.
- const CDSR_OFLOW = CDSR_OFLOW

  DSR flow control of output.
- const CLOCAL = CLOCAL

  ignore modem status lines
- const CREAD = CREAD

  enable receiver
- const CRTSCTS = CRTSCTS

  CTS flow control of output and RTS flow control of input.
- const CRTS_IFLOW = CRTS_IFLOW

  RTS flow control of input.
- const CS5 = CS5

  character size mask: 5 bits
- const CS6 = CS6

  character size mask: 6 bits
- const CS7 = CS7

  character size mask: 7 bits
- const CS8 = CS8

  character size mask: 8 bits
- const CSIZE = CSIZE

  character size mask
- const CSTOPB = CSTOPB

  send 2 stop bits
- const HUPCL = HUPCL

  hang up on last close
- const MDMBUF = MDMBUF

  old name for CCAR_OFLOW
- const PARENB = PARENB

  parity enable
- const PARODD = PARODD

  odd parity, else even
- const OCRNL = OCRNL

  map CR to NL on output
- const ONLCR = ONLCR

  map NL to CR-NL (ala CRMOD)
- const ONLRET = ONLRET

NL performs CR function.

- const ONOCR = ONOCR

  no CR output at column 0
- const ONOEOT = ONOEOT

  discard EOT's (^D) on output
- const OPOST = OPOST

  enable following output processing
- const OXTABS = OXTABS

  expand tabs to spaces
- const BRKINT = BRKINT

  map BREAK to SIGINTR
- const ICRNL = ICRNL

  map CR to NL (ala CRMOD)
- const IGNBRK = IGNBRK

  ignore BREAK condition
- const IGNCR = IGNCR

  ignore CR
- const IGNPAR = IGNPAR

  ignore (discard) parity errors
- const IMAXBEL = IMAXBEL

  ring bell on input queue full
- const INLCR = INLCR

  map NL into CR
- const INPCK = INPCK

  enable checking of parity errors
- const ISTRIP = ISTRIP

  strip 8th bit off chars
- const IXANY = IXANY

  any char will restart after stop
- const IXOFF = IXOFF

  enable input flow control
- const IXON = IXON

  enable output flow control
- const PARMRK = PARMRK

  mark parity and framing errors
- const VDISCARD = VDISCARD

  subscript for the VDISCARD character
- const VDSUSP = VDSUSP

  subscript for the VDSUSP character
- const VEOF = VEOF

  subscript for the EOF character
- const VEOL = VEOL

  subscript for the EOL character
- const VEOL2 = VEOL2

  subscript for the EOL2 character
- const VERASE = VERASE

  subscript for the VERASE character
- const VINTR = VINTR

  subscript for the VINTR character
- const VKILL = VKILL

  subscript for the VKILL character

- const VLNEXT = VLNEXT

  subscript for the VLNEXT character
- const VMIN = VMIN

  subscript for the VMIN value
- const VQUIT = VQUIT

  subscript for the VQUIT character
- const VREPRINT = VREPRINT

  subscript for the VREPRINT character
- const VSTART = VSTART

  subscript for the VSTART character
- const VSTATUS = VSTATUS

  subscript for the character
- const VSTOP = VSTOP

  subscript for the VSTOP character
- const VSUSP = VSUSP

  subscript for the VSUSP character
- const VTIME = VTIME

  subscript for the VTIME value
- const VWERASE = VWERASE

  subscript for the VWERASE character
- const _POSIX_VDISABLE = _POSIX_VDISABLE

  if the value of any key is this value, it means that the key is disabled
- const TCSADRAIN = TCSADRAIN

  drain output, then change
- const TCSAFLUSH = TCSAFLUSH

  drain output, flush input
- const TCSANOW = TCSANOW

  make change immediate
- const TCSASOFT = TCSASOFT

  flag - don't alter hardware state
- const BZ2_DEFAULT_COMPRESSION = BZ2_DEFAULT_COMPRESSION

  gives the default compression level for the bzip2() function, providing maximum compression (value: 9)
- const Z_DEFAULT_COMPRESSION = Z_DEFAULT_COMPRESSION

  gives the default compression level for the compress() and gzip() functions, providing a tradeoff between compression speed and compression size
- const DefaultIV = <0000000000000000>

  The default initialization vector is simply a 8-byte string of nulls.
- const MAXINT = LLONG_MAX

  largest integer
- const MININT = qore(new QoreBigIntNode(-LLONG_MAX - 1))

  smallest integer
- const M_PI = 3.14159265358979323846

  PI (floating-point)
- const M_PIn = qore(pi_number())

  PI (arbitrary-precision numeric)
- const NameToSignal

  maps signal names to signal values
- const SIGABRT = SIGABRT

  SIGABRT.
- const SIGALRM = SIGALRM

  SIGALRM.

- const SIGBUS = SIGBUS

  SIGBUS.
- const SIGCANCEL = SIGCANCEL

  SIGCANCEL.
- const SIGCHLD = SIGCHLD

  SIGCHLD.
- const SIGCLD = SIGCLD

  SIGCLD.
- const SIGCONT = SIGCONT

  SIGCONT.
- const SIGEMT = SIGEMT

  SIGEMT.
- const SIGFPE = SIGFPE

  SIGFPE.
- const SIGFREEZE = SIGFREEZE

  SIGFREEZE.
- const SIGHUP = SIGHUP

  SIGHUP.
- const SIGILL = SIGILL

  SIGILL.
- const SIGINFO = SIGINFO

  SIGINFO.
- const SIGINT = SIGINT

  SIGINT.
- const SIGIO = SIGIO

  SIGIO.
- const SIGIOT = SIGIOT

  SIGIOT.
- const SIGJVM1 = SIGJVM1

  SIGJVM1.
- const SIGJVM2 = SIGJVM2

  SIGJVM2.
- const SIGKILL = SIGKILL

  SIGKILL.
- const SIGLOST = SIGLOST

  SIGLOST.
- const SIGLWP = SIGLWP

  SIGLWP.
- const SIGPIPE = SIGPIPE

  SIGPIPE.
- const SIGPOLL = SIGPOLL

  SIGPOLL.
- const SIGPROF = SIGPROF

  SIGPROF.
- const SIGPWR = SIGPWR

  SIGPWR.
- const SIGQUIT = SIGQUIT

  SIGQUIT.
- const SIGSEGV = SIGSEGV

  SIGSEGV.
- const SIGSTKFLT = SIGSTKFLT

SIGSTKFLT.

- const SIGSTOP = SIGSTOP

  SIGSTOP.
- const SIGSYS = SIGSYS

  SIGSYS.
- const SIGTERM = SIGTERM

  SIGTERM.
- const SIGTHAW = SIGTHAW

  SIGTHAW.
- const SIGTRAP = SIGTRAP

  SIGTRAP.
- const SIGTSTP = SIGTSTP

  SIGTSTP.
- const SIGTTIN = SIGTTIN

  SIGTTIN.
- const SIGTTOU = SIGTTOU

  SIGTTOU.
- const SIGURG = SIGURG

  SIGURG.
- const SIGUSR1 = SIGUSR1

  SIGUSR1.
- const SIGUSR2 = SIGUSR2

  SIGUSR2.
- const SIGVTALRM = SIGVTALRM

  SIGVTALRM.
- const SIGWAITING = SIGWAITING

  SIGWAITING.
- const SIGWINCH = SIGWINCH

  SIGWINCH.
- const SIGXCPU = SIGXCPU

  SIGXCPU.
- const SIGXFSZ = SIGXFSZ

  SIGXFSZ.
- const SIGXRES = SIGXRES

  SIGXRES.
- const SignalToName

  maps signal numbers (as a string key) to the symbolic name for the signal
- const RE_Caseless = PCRE_CASELESS

  Ignores case when matching regular expressions, equivalent to `/i`
- const RE_DotAll = PCRE_DOTALL

  makes a dot (.) match a newline character, equivalent to /s
- const RE_Extended = PCRE_EXTENDED

  ignores whitespace characters and enables comments prefixed by #, equivalent to `/x`
- const RE_Global = QRE_GLOBAL

  replace all matches globally in the string or extract all occurrences of the pattern(s) in the string, equivalent to `/g`
- const RE_MultiLine = PCRE_MULTILINE

  makes start-of-line (^) or end-of-line ($) match after or before any newline in the subject string, equivalent to `/m`

44.1.1 Detailed Description

main Qore-language namespace

Qore namespace.

44.2 Qore::Err Namespace Reference

Qore::Err namespace.

Variables

- const E2BIG = E2BIG

  Argument list too long.
- const EACCES = EACCES

  Permission denied.
- const EADDRINUSE = EADDRINUSE

  Address already in use.
- const EADDRNOTAVAIL = EADDRNOTAVAIL

  Can't assign requested address.
- const EAFNOSUPPORT = EAFNOSUPPORT

  Address family not supported by protocol family.
- const EAGAIN = EAGAIN

  Resource temporarily unavailable.
- const EALREADY = EALREADY

  Operation already in progress.
- const EBADF = EBADF

  Bad file descriptor.
- const EBADMSG = EBADMSG

  Bad message.
- const EBUSY = EBUSY

  Device or Resource busy.
- const ECHILD = ECHILD

  No child processes.
- const ECONNABORTED = ECONNABORTED

  Software caused connection abort.
- const ECONNREFUSED = ECONNREFUSED

  Connection refused.
- const ECONNRESET = ECONNRESET

  Connection reset by peer.
- const EDEADLK = EDEADLK

  Resource deadlock avoided.
- const EDEADLOCK = EDEADLOCK

  Resource deadlock avoided.
- const EDESTADDRREQ = EDESTADDRREQ

  Destination address required.
- const EDOM = EDOM

  Numerical argument out of domain.
- const EDQUOT = EDQUOT

  Disc quota exceeded.

- const EEXIST = EEXIST

  File exists.
- const EFAULT = EFAULT

  Bad address.
- const EFBIG = EFBIG

  File too large.
- const EHOSTDOWN = EHOSTDOWN

  Host is down.
- const EHOSTUNREACH = EHOSTUNREACH

  No route to host.
- const EIDRM = EIDRM

  Identifier removed.
- const EILSEQ = EILSEQ

  Illegal byte sequence.
- const EINPROGRESS = EINPROGRESS

  Operation now in progress.
- const EINTR = EINTR

  Interrupted system call.
- const EINVAL = EINVAL

  Invalid argument.
- const EIO = EIO

  Input/output error.
- const EISCONN = EISCONN

  Socket is already connected.
- const EISDIR = EISDIR

  Is a directory.
- const ELOOP = ELOOP

  Too many levels of symbolic links.
- const EMFILE = EMFILE

  Too many open files.
- const EMLINK = EMLINK

  Too many links.
- const EMSGSIZE = EMSGSIZE

  Message too long.
- const EMULTIHOP = EMULTIHOP

  Reserved.
- const ENAMETOOLONG = ENAMETOOLONG

  File name too long.
- const ENETDOWN = ENETDOWN

  Network is down.
- const ENETRESET = ENETRESET

  Network dropped connection on reset.
- const ENETUNREACH = ENETUNREACH

  Network is unreachable.
- const ENFILE = ENFILE

  Too many open files in system.
- const ENOBUFS = ENOBUFS

  No buffer space available.
- const ENODATA = ENODATA

  No message available on STREAM.
- const ENODEV = ENODEV

Operation not supported by device.

- const ENOENT = ENOENT

  No such file or directory.
- const ENOEXEC = ENOEXEC

  Exec format error.
- const ENOLCK = ENOLCK

  No locks available.
- const ENOLINK = ENOLINK

  Reserved.
- const ENOMEM = ENOMEM

  Cannot allocate memory.
- const ENOMSG = ENOMSG

  No message of desired type.
- const ENOPROTOOPT = ENOPROTOOPT

  Protocol not available.
- const ENOSPC = ENOSPC

  No space left on device.
- const ENOSR = ENOSR

  No STREAM resources.
- const ENOSTR = ENOSTR

  Not a STREAM.
- const ENOSYS = ENOSYS

  Function not implemented.
- const ENOTBLK = ENOTBLK

  Block device required.
- const ENOTCONN = ENOTCONN

  Socket is not connected.
- const ENOTDIR = ENOTDIR

  Not a directory.
- const ENOTEMPTY = ENOTEMPTY

  Directory not empty.
- const ENOTSOCK = ENOTSOCK

  Socket operation on non-socket.
- const ENOTTY = ENOTTY

  Inappropriate ioctl for device.
- const ENXIO = ENXIO

  Device not configured.
- const EOPNOTSUPP = EOPNOTSUPP

  Operation not supported on socket.
- const EOVERFLOW = EOVERFLOW

  Value too large to be stored in data type.
- const EPERM = EPERM

  Operation not permitted.
- const EPFNOSUPPORT = EPFNOSUPPORT

  Protocol family not supported.
- const EPIPE = EPIPE

  Broken pipe.
- const EPROTO = EPROTO

  Protocol error.
- const EPROTONOSUPPORT = EPROTONOSUPPORT

  Protocol not supported.

- const EPROTOTYPE = EPROTOTYPE

  Protocol wrong type for socket.
- const ERANGE = ERANGE

  Result too large.
- const EREMOTE = EREMOTE

  Too many levels of remote in path.
- const EROFS = EROFS

  Read-only file system.
- const ESHUTDOWN = ESHUTDOWN

  Can't send after socket shutdown.
- const ESOCKTNOSUPPORT = ESOCKTNOSUPPORT

  Socket type not supported.
- const ESRCH = ESRCH

  search error
- const ESTALE = ESTALE

  Stale NFS file handle.
- const ETIME = ETIME

  STREAM ioctl timeout.
- const ETIMEDOUT = ETIMEDOUT

  Operation timed out.
- const ETOOMANYREFS = ETOOMANYREFS

  Too many references: can't splice.
- const ETXTBSY = ETXTBSY

  Text file busy.
- const EUSERS = EUSERS

  Too many users.
- const EWOULDBLOCK = EWOULDBLOCK

  Operation would block.
- const EXDEV = EXDEV

  Cross-device link.

44.2.1 Detailed Description

Qore::Err namespace.

44.3 Qore::Option Namespace Reference

Qore::Option namespace.

Variables

- const HAVE_ATOMIC_OPERATIONS = bool(QORE_CONST_HAVE_ATOMIC_MACROS)

  Indicates if the Qore library supports fast atomic reference counting.
- const HAVE_CLOSE_ALL_FD = bool(QORE_CONST_HAVE_CLOSE_ALL_FD)

  Indicates if the close_all_fd() function is available.
- const HAVE_FILE_LOCKING = bool(QORE_CONST_HAVE_STRUCT_FLOCK)

  Indicates if the Qore library supports file locking; currently this depends on UNIX-style file locking with the fnctl() function.
- const HAVE_FORK = bool(QORE_CONST_HAVE_FORK)

Indicates if the fork() function is available.

- const HAVE_GETPPID = bool(QORE_CONST_HAVE_GETPPID)

  Indicates if the getppid() function is available.
- const HAVE_IS_EXECUTABLE = bool(QORE_CONST_HAVE_PWD_H)

  Indicates if the Qore library supports the is_executable() function.
- const HAVE_KILL = bool(QORE_CONST_HAVE_KILL)

  Indicates if the kill() function is available.
- const HAVE_LIBRARY_DEBUGGING = bool(QORE_CONST_DEBUG)

  Indicates if the Qore library has been built with debugging enabled.
- const HAVE_MD2 = bool(QORE_CONST_HAVE_MD2)

  Indicates if the openssl library used to build the qore library supported the MD2 algorithm and therefore if the MD2() and MD2_bin() functions are available.
- const HAVE_MDC2 = bool(QORE_CONST_HAVE_MDC2)

  Indicates if the openssl library used to build the qore library supported the MDC2 algorithm and therefore if the MDC2() and MDC2_bin() functions are available.
- const HAVE_RC5 = bool(QORE_CONST_HAVE_RC5)

  Indicates if the openssl library used to build the qore library supported the RC5 encryption algorithm and therefore if the rc5_encrypt_cbc(), rc5_decrypt_cbc() and rc5_encrypt_cbc_to_string() functions are available.
- const HAVE_ROUND = bool(QORE_CONST_HAVE_ROUND)

  Indicates if the round() function is available; the availability of this function depends on the presence of the C-library's round() function.
- const HAVE_RUNTIME_THREAD_STACK_TRACE = bool(QORE_CONST_QORE_RUNTIME_THREAD↩ _STACK_TRACE)

  Indicates if active thread stack tracing has been enabled as a debugging option and if the getAllThreadCallStacks() function is available.
- const HAVE_SETEGID = bool(QORE_CONST_HAVE_SETEGID)

  Indicates if the setegid() function is available; the availabilty of this function depends on the system's underlying C-library.
- const HAVE_SETEUID = bool(QORE_CONST_HAVE_SETEUID)

  Indicates if the seteuid() function is available; the availabilty of this function depends on the system's underlying C-library.
- const HAVE_SETSID = bool(QORE_CONST_HAVE_SETSID)

  Indicates if the setsid() function is available.
- const HAVE_SHA224 = bool(QORE_CONST_HAVE_SHA256)

  Indicates if the openssl library used to build the qore library supported the SHA224 algorithm and therefore if the SHA224() and SHA224_bin() functions are available.
- const HAVE_SHA256 = bool(QORE_CONST_HAVE_SHA256)

  Indicates if the openssl library used to build the qore library supported the SHA256 algorithm and therefore if the SHA256() and SHA256_bin() functions are available.
- const HAVE_SHA384 = bool(QORE_CONST_HAVE_SHA512)

  Indicates if the openssl library used to build the qore library supported the SHA384 algorithm and therefore if the SHA384() and SHA384_bin() functions are available.
- const HAVE_SHA512 = bool(QORE_CONST_HAVE_SHA512)

  Indicates if the openssl library used to build the qore library supported the SHA512 algorithm and therefore if the SHA512() and SHA512_bin() functions are available.
- const HAVE_SIGNAL_HANDLING = qore(get_bool_node(QORE_CONST_HAVE_SIGNAL_HANDLING && !(qore_library_options & QLO_DISABLE_SIGNAL_HANDLING)))

  Indicates if UNIX-style signal handling is available.
- const HAVE_STACK_GUARD = bool(QORE_CONST_HAVE_CHECK_STACK_POS)

  Indicates if protection against stack overruns is provided.
- const HAVE_STATVFS = bool(QORE_CONST_HAVE_SYS_STATVFS_H)

  Indicates if the statvfs() function is available.
- const HAVE_SYMLINK = bool(QORE_CONST_HAVE_SYMLINK)

Indicates if the symlink() function is available.

- const HAVE_SYSTEM = bool(QORE_CONST_HAVE_SYSTEM)

  Indicates if the system() function is available.
- const HAVE_TERMIOS = bool(QORE_CONST_HAVE_TERMIOS_H)

  Indicates if the TermIOS class is available.
- const HAVE_UNIX_FILEMGT = bool(QORE_CONST_HAVE_CHOWN)

  Indicates if UNIX-style file management functionality is available (ex: chown(), Dir::chgrp(), etc)
- const HAVE_UNIX_USERMGT = bool(QORE_CONST_HAVE_GETUID)

  Indicates if UNIX-style user management functionality is available (ex: getuid(), setuid(), getgid(), setgid(), etc)

44.3.1 Detailed Description

Qore::Option namespace.

44.4 Qore::SQL Namespace Reference

Qore::SQL namespace.

Classes

- class AbstractDatasource

  This class defines an abstract interface for database access, inherited by both the Datasource and DatasourcePool classes.
- class Datasource

  This class provides the Qore interface to databases.
- class DatasourcePool

  Provides transparent per-thread, per-transaction datasource connection pooling.
- class SQLStatement

  The SQLStatement class provides the most flexibilty for executing SQL on a database server.

Functions

- __7_ int getDBIDriverCapabilities (string driver)

  Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or NOTHING if the driver is not already loaded.
- nothing getDBIDriverCapabilities ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ list getDBIDriverCapabilityList (string driver)

  Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.
- nothing getDBIDriverCapabilityList ()

  This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- __7_ list getDBIDriverList ()

  Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.
- hash parseDatasource (string ds)

  Returns a datasource hash of the components of a datasource string.
- nothing parseDatasource ()

This function variant does nothing at all; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

- int dbi_get_driver_capabilities (string driver)

  Returns an integer representing the capabilities of a DBI driver binary-OR'ed together (see DBI Capability Constants) or 0 if the driver is not already loaded.

- __7_ list dbi_get_driver_capability_list (string driver)

  Returns a list of each capability supported by the given DBI driver (see DBI Capability Constants) or NOTHING if the driver cannot be found.

- __7_ list dbi_get_driver_list ()

  Returns a list of strings of DBI drivers currently loaded or NOTHING if no drivers are loaded.

- __7_ hash dbi_get_driver_options (string driver)

  returns a hash of driver options

- hash parse_datasource (string ds)

  Returns a datasource hash of the components of a datasource string.

Variables

- const DSDB2 = "db2"

  for the "db2" driver

- const DSFreeTDS = "freetds"

  for the "freetds" driver

- const DSMSSQL = "freetds"

  another constant for the "freetds" driver

- const DSMySQL = "mysql"

  for the "mysql" driver

- const DSOracle = "oracle"

  for the "oracle" driver

- const DSPGSQL = "pgsql"

  for the "pgsql" driver

- const DSSQLite3 = "sqlite3"

  for the "sqlite3" driver

- const DSSybase = "sybase"

  for the "sybase" driver

- const DBI_CAP_AUTORECONNECT = DBI_CAP_AUTORECONNECT

  Indicates that the DBI driver supports automatically/transparently reconnecting to the server if the connection is lost while not in a transaction.

- const DBI_CAP_BIND_BY_PLACEHOLDER = DBI_CAP_BIND_BY_PLACEHOLDER

  Indicates that the DBI driver supports binding placeholder buffers when executing SQL to retrieve data from queries and procedures, etc.

- const DBI_CAP_BIND_BY_VALUE = DBI_CAP_BIND_BY_VALUE

  Indicates that the DBI driver supports directly binding qore values into queries using the %v placeholder in the query string.

- const DBI_CAP_CHARSET_SUPPORT = DBI_CAP_CHARSET_SUPPORT

  Indicates that the DBI driver supports proper character encoding conversions.

- const DBI_CAP_EVENTS = DBI_CAP_EVENTS

  Indicates that the DBI driver supports the event API.

- const DBI_CAP_HAS_DESCRIBE = DBI_CAP_HAS_DESCRIBE

  Indicates that the DBI driver supports the describe method.

- const DBI_CAP_HAS_EXECRAW = DBI_CAP_HAS_EXECRAW

  Indicates that the DBI driver supports the Datasource::execRaw() and DatasourcePool::execRaw() methods.

- const DBI_CAP_HAS_NUMBER_SUPPORT = DBI_CAP_HAS_NUMBER_SUPPORT

  Indicates that the DBI driver supports arbitrary-precision numeric support for binding and retrieving values.

- const DBI_CAP_HAS_OPTION_SUPPORT = DBI_CAP_HAS_OPTION_SUPPORT

  Indicates that the DBI driver supports the new driver option API.
- const DBI_CAP_HAS_SELECT_ROW = DBI_CAP_HAS_SELECT_ROW

  Indicates that the DBI driver supports a native selectRow() method implementation.
- const DBI_CAP_HAS_STATEMENT = DBI_CAP_HAS_STATEMENT

  Indicates that the DBI driver supports the prepared statement interface (the SQLStatement class)
- const DBI_CAP_LOB_SUPPORT = DBI_CAP_LOB_SUPPORT

  Indicates that the DBI driver supports LOB columns (BLOBs and CLOBs, for example)
- const DBI_CAP_SERVER_TIME_ZONE = DBI_CAP_SERVER_TIME_ZONE

  Indicates that the DBI driver supports automatically converting date/time values to the server's presumed time zone (also implies that the driver supports the "timezone" option) and tagging date/time values with the same; this is independent from the client's current time zone setting.
- const DBI_CAP_STORED_PROCEDURES = DBI_CAP_STORED_PROCEDURES

  Indicates that the DBI driver supports stored procedure execution.
- const DBI_CAP_TIME_ZONE_SUPPORT = DBI_CAP_TIME_ZONE_SUPPORT

  Indicates that the DBI driver supports time zones in times.
- const DBI_CAP_TRANSACTION_MANAGEMENT = DBI_CAP_TRANSACTION_MANAGEMENT

  Indicates that the DBI driver supports transaction management.
- const BLOB = "blob"

  for binding BLOB values
- const CLOB = "clob"

  for binding CLOB values
- const DATE = "date"

  fod binding date/time values
- const DECIMAL = "number"

  for binding decimal values as a number
- const NUMBER = "number"

  for binding number values as a number
- const NUMERIC = "number"

  for binding numeric values as a number
- const VARCHAR = "string"

  for binding string values

44.4.1 Detailed Description

Qore::SQL namespace.

44.5 Qore::Thread Namespace Reference

Qore::Thread namespace.

Classes

- class AbstractSmartLock

  The abstract base class for locks that support the internal API for use with the Condition class.
- class AutoGate

  A helper class for the Gate class for exception-safe Gate handling.
- class AutoLock

  A helper class for the Mutex class for exception-safe Mutex handling.

- class AutoReadLock

  A helper class for the RWLock class for exception-safe read lock handling.
- class AutoWriteLock

  A helper class for the RWLock class for exception-safe write lock handling.
- class Condition

  The Condition class can be used For blocking a thread until a condition becomes True.
- class Counter

  Implements a class that can be used for blocking a thread until a counter reaches zero.
- class Gate

  The Gate class implements a reentrant thread lock.
- class Mutex

  A class providing an implementation for a simple thread lock.
- class Queue

  Queue objects provide a blocking, thread-safe message-passing object to Qore programs
- class RWLock

  The RWLock class implements a read-write thread lock.
- class Sequence

  The Sequence class implements a thread-safe increment-only object.
- class ThreadPool

  This class defines a thread pool that grows and shrinks dynamically within user-defined limits according to the task load placed on it.

44.5.1 Detailed Description

Qore::Thread namespace.

44.6 Qore::Type Namespace Reference

Qore::Type namespace.

Variables

- const Binary = "binary"

  Gives the type for binary values.
- const Boolean = "bool"

  Gives the type for boolean values.
- const CallReference = "call reference"

  Gives the type for call references.
- const Closure = "closure"

  Gives the type for closures.
- const Date = "date"

  Gives the type for the date values.
- const Float = "float"

  Gives the type for float values.
- const Hash = "hash"

  Gives the type for hash values.
- const Int = "integer"

  Gives the type for integer values.
- const List = "list"

Gives the type for list values.

- const NothingType = "nothing"

  Gives the type when no value is available.

- const NullType = "NULL"

  Gives the type for SQL null values.

- const Number = "number"

  Gives the type for number values.

- const Object = "object"

  Gives the type for object values.

- const String = "string"

  Gives the type for string values.

44.6.1 Detailed Description

Qore::Type namespace.

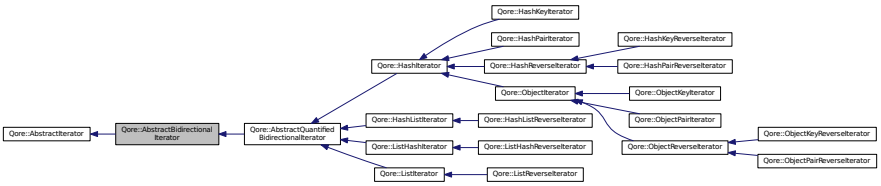
Chapter 45

Class Documentation

45.1 Qore::AbstractBidirectionalIterator Class Reference

This class defines an abstract interface for bidirectional iterators.

Inheritance diagram for Qore::AbstractBidirectionalIterator:



Public Member Functions

- abstract bool prev ()

  Moves the current position to the previous element; returns False if there are no more elements.

45.1.1 Detailed Description

This class defines an abstract interface for bidirectional iterators.

45.1.2 Member Function Documentation

45.1.2.1 abstract bool Qore::AbstractBidirectionalIterator::prev ( ) [pure virtual]

Moves the current position to the previous element; returns False if there are no more elements.

This method will return True again after it returns False once if the object being iterated is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
```

```
        printf(" + %y\n", $i.getValue());
    }
```

Note

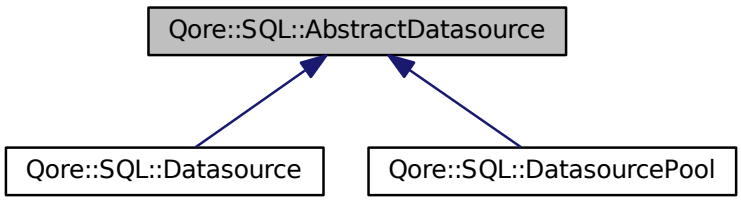
this method is basically the opposite to AbstractIterator::next()

Implemented in Qore::HashListIterator, Qore::ListHashIterator, Qore::HashIterator, Qore::ListIterator, Qore::HashListReverseIterator, Qore::ListHashReverseIterator, Qore::ObjectReverseIterator, Qore::HashReverseIterator, and Qore::ListReverseIterator.

45.2 Qore::SQL::AbstractDatasource Class Reference

This class defines an abstract interface for database access, inherited by both the Datasource and DatasourcePool classes.

Inheritance diagram for Qore::SQL::AbstractDatasource:



Public Member Functions

- abstract nothing beginTransaction ()

  Manually signals the start of transaction management on the AbstractDatasource.
- abstract nothing commit ()

  Commits the current transaction and releases any thread resources associated with the transaction.
- bool currentThreadInTransaction ()

  Should return True if the current thread is in a transaction with this object, must be re-implemented in subclasses to provide the desired functionality.
- abstract any exec (string sql,...)

  Executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values)
- abstract any execRaw (string sql)

  Executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)
- abstract any getClientVersion ()

  Retrieves the driver-specific client library version information.
- abstract hash getConfigHash ()

  Returns a datasource hash describing the configuration of the current object.
- abstract string getConfigString ()

  Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()
- abstract string getDBEncoding ()

Retrieves the database-specific charset set encoding for the object.

- abstract __7_ string getDBName ()

  Returns the database name parameter as a string or NOTHING if none is set.

- abstract string getDriverName ()

  Returns the name of the driver used for the object.

- abstract __7_ string getHostName ()

  Returns the hostname parameter as a string or NOTHING if none is set.

- abstract __7_ string getOSEncoding ()

  Returns the Qore character encoding name for the object as a string or NOTHING if none is set.

- abstract __7_ string getPassword ()

  Returns the password parameter as a string or NOTHING if none is set.

- abstract __7_ int getPort ()

  Gets the port number that will be used for the next connection to the server.

- abstract any getServerVersion ()

  Returns the driver-specific server version data for the current connection.

- abstract __7_ string getUserName ()

  Returns the username parameter as a string or NOTHING if none is set.

- abstract bool inTransaction ()

  Returns True if a transaction is currently in progress.

- abstract nothing rollback ()

  Rolls the current transaction back and releases any thread resources associated with the transaction.

- abstract any select (string sql,...)

  Executes an SQL select statement on the server and (normally) returns the result as a hash (column names) of lists (column values per row)

- abstract any selectRow (string sql,...)

  Executes an SQL select statement on the server and returns the first row as a hash (the column values)

- abstract any selectRows (string sql,...)

  Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

- abstract any vexec (string sql, __7_ softlist vargs)

  Executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values), taking a list for all bind arguments.

- abstract any vselect (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.

- abstract any vselectRow (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.

- abstract any vselectRows (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

45.2.1 Detailed Description

This class defines an abstract interface for database access, inherited by both the Datasource and DatasourcePool classes.

Restrictions:

Qore::PO_NO_DATABASE

45.2.2 Member Function Documentation

45.2.2.1 abstract nothing Qore::SQL::AbstractDatasource::beginTransaction ( ) [pure virtual]

Manually signals the start of transaction management on the AbstractDatasource.

This method should be called when the AbstractDatasource object will be shared between more than 1 thread, and a transaction will be started with a AbstractDatasource::select() method or the like.

This method does not make any communication with the server to start a transaction; it only allocates the transaction lock to the current thread in Qore.

Example:

```
$db.beginTransaction();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.2 abstract nothing Qore::SQL::AbstractDatasource::commit ( ) [pure virtual]

Commits the current transaction and releases any thread resources associated with the transaction.

Example:

```
$db.commit();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.3 bool Qore::SQL::AbstractDatasource::currentThreadInTransaction ( )

Should return True if the current thread is in a transaction with this object, must be re-implemented in subclasses to provide the desired functionality.

Note

- this is reimplemented as Datasource::currentThreadInTransaction() and DatasourcePool::currentThreadInTransaction()
- this method was added as a non-abstract method in Qore 0.8.10 to avoid breaking existing subclasses of AbstractDatasource

Since

Qore 0.8.10

45.2.2.4 abstract any Qore::SQL::AbstractDatasource::exec (string *sql*, ...) [pure virtual]

Executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values)

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.exec("insert into table (varchar_col, timestamp_col, blob_col, numeric_col) values (%v,
       %v, %v, %d)", $string, now(), $binary, 100);
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.5 abstract any Qore::SQL::AbstractDatasource::execRaw (string *sql*) [pure virtual]

Executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)

This method does not do any variable binding, so it's useful for example for DDL statements etc

Warning:

Using this method to execute pure dynamic SQL many times with different SQL strings (as opposed to using the same string and binding by value instead of dynamic SQL) can affect application performance by prohibiting the efficient usage of the DB server's statement cache. See DB server documentation for variable binding and the SQL statement cache for more information.

Parameters

sql	The SQL command to execute on the server; this string will not be subjected to any transformations for variable binding

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
$db.execRaw("create table my_tab (id number, some_text varchar2(30))");
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.6 abstract any Qore::SQL::AbstractDatasource::getClientVersion ( ) [pure virtual]

Retrieves the driver-specific client library version information.

Returns

the driver-specific client library version information

Example:

```
my any $ver = $db.getClientVersion();
```

Note

see the documentation for the DBI driver being used for possible exceptions

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.7 abstract hash Qore::SQL::AbstractDatasource::getConfigHash ( ) [pure virtual]

Returns a datasource hash describing the configuration of the current object.

Example:

```
my hash $h = $ds.getConfigHash();
```

Returns

a datasource hash describing the configuration of the current object

Since

Qore 0.8.8

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.8 abstract string Qore::SQL::AbstractDatasource::getConfigString ( ) [pure virtual]

Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Example:

```
my string $str = $ds.getConfigString();
```

Returns

a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Since

Qore 0.8.8

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.9 abstract string Qore::SQL::AbstractDatasource::getDBEncoding ( ) [pure virtual]

Retrieves the database-specific charset set encoding for the object.

Returns

the database-specific charset set encoding for the object

Example:

```
my string $enc = $db.getDBEncoding();
```

See also

AbstractDatasource::getOSEncoding();

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.10 abstract __7_ string Qore::SQL::AbstractDatasource::getDBName ( ) [pure virtual]

Returns the database name parameter as a string or NOTHING if none is set.

Returns

the database name parameter as a string or NOTHING if none is set

Example:

```
my *string $db = $db.getDBName();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.11 abstract string Qore::SQL::AbstractDatasource::getDriverName ( ) [pure virtual]

Returns the name of the driver used for the object.

Returns

the name of the driver used for the object

Example:

```
my string $driver = $db.getDriverName();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.12 abstract __7_ string Qore::SQL::AbstractDatasource::getHostName ( ) [pure virtual]

Returns the hostname parameter as a string or NOTHING if none is set.

Returns

the hostname parameter as a string or NOTHING if none is set

Example:

```
my *string $host = $db.getHostName();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.13 abstract __7_ string Qore::SQL::AbstractDatasource::getOSEncoding ( ) [pure virtual]

Returns the Qore character encoding name for the object as a string or NOTHING if none is set.

Returns

the Qore character encoding name for the object as a string or NOTHING if none is set

Example:

```
my *string $enc = $db.getOSEncoding();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.14 abstract __7_ string Qore::SQL::AbstractDatasource::getPassword ( ) [pure virtual]

Returns the password parameter as a string or NOTHING if none is set.

Returns

the password parameter as a string or NOTHING if none is set

Example:

```
my *string $pass = $db.getPassword();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.15 abstract __7_ int Qore::SQL::AbstractDatasource::getPort ( ) [pure virtual]

Gets the port number that will be used for the next connection to the server.

Invalid port numbers will cause an exception to be thrown when the connection is opened

Example:

```
my *int $port = $db.getPort();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.16 abstract any Qore::SQL::AbstractDatasource::getServerVersion ( ) [pure virtual]

Returns the driver-specific server version data for the current connection.

Returns

the driver-specific server version data for the current connection

Example:

```
my any $ver = $db.getServerVersion();
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.17 abstract __7_ string Qore::SQL::AbstractDatasource::getUserName ( ) [pure virtual]

Returns the username parameter as a string or NOTHING if none is set.

Returns

the username parameter as a string or NOTHING if none is set

Example:

```
my *string $user = $db.getUserName();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.18 abstract bool Qore::SQL::AbstractDatasource::inTransaction ( ) [pure virtual]

Returns True if a transaction is currently in progress.

Returns

True if a transaction is currently in progress

Example:

```
my bool $b = $db.inTransaction();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.19 abstract nothing Qore::SQL::AbstractDatasource::rollback ( ) [pure virtual]

Rolls the current transaction back and releases any thread resources associated with the transaction.

Example:

```
$db.rollback();
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.20 abstract any Qore::SQL::AbstractDatasource::select (string *sql*, ...) [pure virtual]

Executes an SQL select statement on the server and (normally) returns the result as a hash (column names) of lists (column values per row)

The usual return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

Additionally, this format is a more efficient format than that returned by the AbstractDatasource::selectRows() method, because the column names are not repeated for each row returned. Therefore, for retrieving anything greater than small amounts of data, it is recommended to use this method instead of AbstractDatasource::selectRows().

To execute select statements that begin a transaction (such as "`select for update`"), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method returns a hash (the keys are the column names) of lists (the column data per row) when executed with an SQL select statement, however some DBI drivers allow any SQL to be executed through this method, in which case other data types can be returned (such as an integer for a row count or a hash for output parameters when executing a stored procedure)

Example:

```
# bind a string and a date/time value by value in a query
$query = $db.select("select * from table where varchar_column = %v and timestamp_column > %v", $string,
       2007-10-11T15:31:26.289);
```

Note

This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.21 abstract any Qore::SQL::AbstractDatasource::selectRow (string *sql*, ...) [pure virtual]

Executes an SQL select statement on the server and returns the first row as a hash (the column values)

If more than one row is returned, then it is treated as an error and a `DBI-SELECT-ROW-ERROR` is returned (however the DBI driver should raise its own exception here to avoid retrieving more than one row from the server). For a similar method taking a list for all bind arguments, see AbstractDatasource::vselectRow().

This method also accepts all bind parameters (`d`, `v`, etc) as documented in Binding by Value and Placeholder

To execute select statements that begin a transaction (such as `"select for update"`), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method normally returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $h = $db.selectRow("select * from example_table where id = 1");
```

Exceptions

DBI-SELECT-ROW-ERR OR	more than 1 row retrieved from the server

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.22 abstract any Qore::SQL::AbstractDatasource::selectRows (string *sql*, ...) [pure virtual]

Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

The return format of this method is not as memory efficient as that returned by the AbstractDatasource::select() method, therefore for larger amounts of data, it is recommended to use AbstractDatasource::select(). The usual return value of this method can be iterated with the ListHashIterator class.

This method also accepts all bind parameters (`d`, `v`, etc) as documented in Binding by Value and Placeholder

To execute select statements that begin a transaction (such as `"select for update"`), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.selectRows("select * from example_table");
```

See also

AbstractDatasource::select()

Note

This method returns all the data available immediately; to process query data piecewise, use the SQL↩ Statement class

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.23 abstract any Qore::SQL::AbstractDatasource::vexec (string *sql*, __7_ softlist *vargs*) [pure virtual]

Executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values), taking a list for all bind arguments.

Same as AbstractDatasource::exec() except takes an explicit list for bind arguments

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.vexec("insert into example_table value (%v, %v, %v)", $arg_list);
```

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.24 abstract any Qore::SQL::AbstractDatasource::vselect (string *sql*, __7_ softlist *vargs*) [pure virtual]

Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.

The usual return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

This method also accepts all bind parameters (`d`, `v`, etc) as documented in Binding by Value and Placeholder

To execute select statements that begin a transaction (such as `"select for update"`), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a hash (the keys are the column names) of list (each hash key's value is a list giving the row data), however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $query = $db.vselect("select * from example_table where id = %v and name = %v", $arg_list);
```

See also

AbstractDatasource::select()

Note

This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.25 abstract any Qore::SQL::AbstractDatasource::vselectRow (string *sql*, __7_ softlist *vargs*) [pure virtual]

Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.

This method is the same as the AbstractDatasource::selectRow() method, except this method takes a single argument after the SQL command giving the list of bind value parameters

This method also accepts all bind parameters (`d`, `v`, etc) as documented in Binding by Value and Placeholder

To execute select statements that begin a transaction (such as `"select for update"`), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method normally returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $h = $db.vselectRow("select * from example_table where id = %v and type = %v", $arg_list);
```

See also

AbstractDatasource::selectRow()

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.2.2.26 abstract any Qore::SQL::AbstractDatasource::vselectRows (string *sql*, _7_ softlist *vargs*) [pure virtual]

Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

Same as the AbstractDatasource::selectRows() method, except this method takes a single argument after the SQL command giving the list of bind value parameters.

The usual return value of this method can be iterated with the ListHashIterator class.

The return format of this method is not as memory efficient as that returned by the AbstractDatasource::select() method, therefore for larger amounts of data, it is recommended to use AbstractDatasource::select().

This method also accepts all bind parameters (`d`, `v`, etc) as documented in Binding by Value and Placeholder

To execute select statements that begin a transaction (such as `"select for update"`), execute AbstractDatasource::beginTransaction() first to signal that a transaction is starting; this is particularly important when the object is shared among more than one thread.

Parameters

sql	The SQL command to execute
vargs	Include any values to be bound (using `v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.vselectRows("select * from example_table where id = %v and type = %v", $arg_list);
```

See also

AbstractDatasource::selectRows()

Note

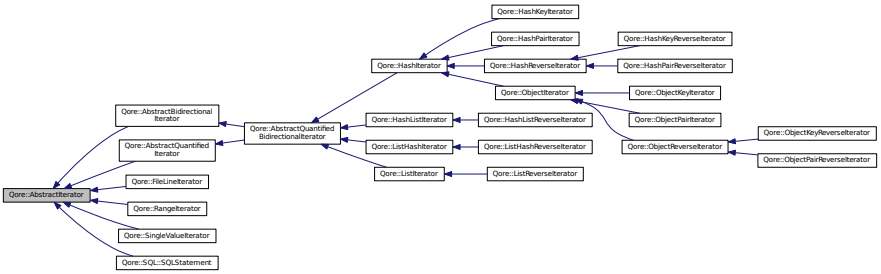
This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

Implemented in Qore::SQL::Datasource, and Qore::SQL::DatasourcePool.

45.3 Qore::AbstractIterator Class Reference

This class defines an abstract interface for iterators.

Inheritance diagram for Qore::AbstractIterator:



Public Member Functions

- abstract any getValue ()

  returns the current value
- abstract bool next ()

  Moves the current position to the next element; returns False if there are no more elements.
- abstract bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.3.1 Detailed Description

This class defines an abstract interface for iterators.

Classes inheriting this class can be used to iterate abstract or complex objects with while Statements (using AbstractIterator::next()) or directly in the following language constructs:

- Map Operator (map)
- Fold Right Operator (foldr) and Fold Left Operator (foldl)
- Select From List Operator (select)
- foreach Statements

45.3.2 Member Function Documentation

45.3.2.1 abstract any Qore::AbstractIterator::getValue ( ) [pure virtual]

returns the current value

Returns

the current value

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element

See also

AbstractIterator::valid()

Implemented in Qore::SQL::SQLStatement, Qore::HashListIterator, Qore::HashIterator, Qore::ListHashIterator, Qore::FileLineIterator, Qore::ListIterator, Qore::ObjectPairReverseIterator, Qore::ObjectKeyReverseIterator, Qore::HashKeyReverseIterator, Qore::HashPairReverseIterator, Qore::ObjectPairIterator, Qore::ObjectKeyIterator, Qore::HashPairIterator, Qore::HashKeyIterator, Qore::RangeIterator, and Qore::SingleValueIterator.

45.3.2.2 abstract bool Qore::AbstractIterator::next ( ) [pure virtual]

Moves the current position to the next element; returns False if there are no more elements.

This method will return True again after it returns False once if the object being iterated is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y\n", $i.getValue());
}
```

Implemented in Qore::SQL::SQLStatement, Qore::HashListIterator, Qore::ListHashIterator, Qore::HashIterator, Qore::FileLineIterator, Qore::ListIterator, Qore::HashListReverseIterator, Qore::ListHashReverseIterator, Qore::ObjectReverseIterator, Qore::HashReverseIterator, Qore::ListReverseIterator, Qore::RangeIterator, and Qore::SingleValueIterator.

45.3.2.3 abstract bool Qore::AbstractIterator::valid ( ) [pure virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Example:

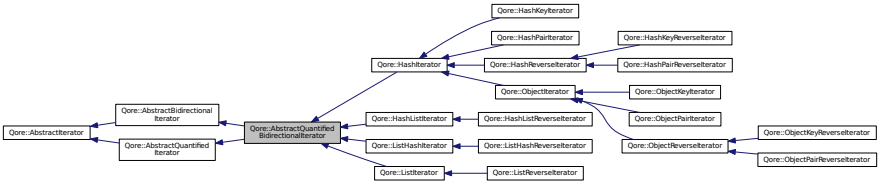
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implemented in Qore::SQL::SQLStatement, Qore::HashListIterator, Qore::ListHashIterator, Qore::HashIterator, Qore::ListIterator, Qore::FileLineIterator, Qore::RangeIterator, and Qore::SingleValueIterator.

45.4 Qore::AbstractQuantifiedBidirectionalIterator Class Reference

This class defines an abstract interface for bidirectional iterators where the size of the object is known in advance.

Inheritance diagram for Qore::AbstractQuantifiedBidirectionalIterator:



Additional Inherited Members

45.4.1 Detailed Description

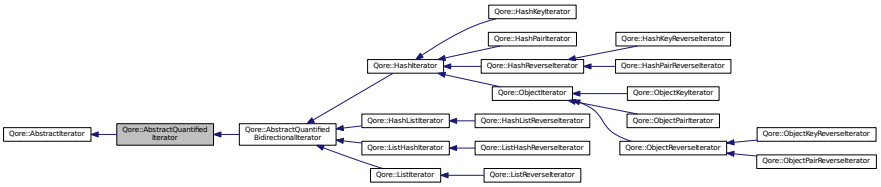
This class defines an abstract interface for bidirectional iterators where the size of the object is known in advance.

This class does not define any new methods but rather just combines two abstract interfaces into one.

45.5 Qore::AbstractQuantifiedIterator Class Reference

This class defines an abstract interface for iterators where the size of the object being iterated is known in advance.

Inheritance diagram for Qore::AbstractQuantifiedIterator:



Public Member Functions

- abstract bool empty ()

  returns True if the object to iterate is empty; False if not
- abstract bool first ()

  returns True if on the first element
- abstract bool last ()

  returns True if on the last element

45.5.1 Detailed Description

This class defines an abstract interface for iterators where the size of the object being iterated is known in advance.

45.5.2 Member Function Documentation

45.5.2.1 abstract bool Qore::AbstractQuantifiedIterator::empty ( ) [pure virtual]

returns True if the object to iterate is empty; False if not

Returns

True if the object to iterate is empty; False if not

Example:

```
if ($i.empty())
    printf("object is empty\n");
```

Implemented in Qore::HashListIterator, Qore::HashIterator, Qore::ListHashIterator, and Qore::ListIterator.

45.5.2.2 abstract bool Qore::AbstractQuantifiedIterator::first ( ) [pure virtual]

returns True if on the first element

Returns

True if on the first element

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Implemented in Qore::HashListIterator, Qore::HashIterator, Qore::ObjectReverseIterator, Qore::HashListReverseIterator, Qore::ListHashIterator, Qore::ListIterator, Qore::HashReverseIterator, Qore::ListHashReverseIterator, and Qore::ListReverseIterator.

45.5.2.3 abstract bool Qore::AbstractQuantifiedIterator::last ( ) [pure virtual]

returns True if on the last element

Returns

True if on the last element

Example:

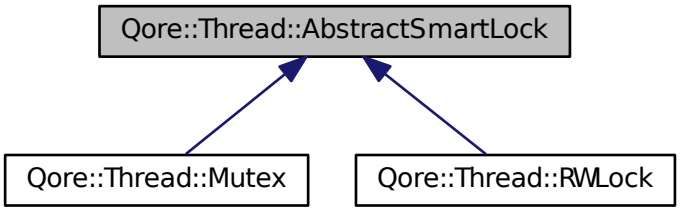
```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Implemented in Qore::HashListIterator, Qore::HashIterator, Qore::ListHashIterator, Qore::ListIterator, Qore::ObjectReverseIterator, Qore::HashListReverseIterator, Qore::HashReverseIterator, Qore::ListHashReverseIterator, and Qore::ListReverseIterator.

45.6 Qore::Thread::AbstractSmartLock Class Reference

The abstract base class for locks that support the internal API for use with the Condition class.

Inheritance diagram for Qore::Thread::AbstractSmartLock:



Public Member Functions

- constructor ()

  Throws an exception if called directly; this class can only be instantiated by builtin subclasses.
- string getName ()

  Returns the name of the threading class directly inheriting this class.
- bool lockOwner ()

  Returns True if the calling thread owns the lock, False if not.
- int lockTID ()

  Returns the TID of the thread owning the lock or -1 if the lock is currently not acquired.

45.6.1 Detailed Description

The abstract base class for locks that support the internal API for use with the Condition class.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

This is an abstract class to be inherited by builtin classes that implement the internal Qore API that allows them to be used by the Condition class. Currently the RWLock and Mutex classes inherit this class.

This class cannot be instantiated directly and also cannot be directly inherited by user-defined classes.

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.6.2 Member Function Documentation

45.6.2.1 Qore::Thread::AbstractSmartLock::constructor ( )

Throws an exception if called directly; this class can only be instantiated by builtin subclasses.

Exceptions

ABSTRACTSMARTLOCK-CONSTRUCTOR-ERROR	this exception is thrown if this class is constructed directly (also if directly inherited by user classes)

45.6.2.2 string Qore::Thread::AbstractSmartLock::getName ( )

Returns the name of the threading class directly inheriting this class.

Code Flags:

CONSTANT

Example:

```
my string $name = $lock.getName();
```

Returns

the name of the threading class directly inheriting this class

45.6.2.3 bool Qore::Thread::AbstractSmartLock::lockOwner ( )

Returns True if the calling thread owns the lock, False if not.

Code Flags:

CONSTANT

Example:

```
# only grab and release lock if we don't already own it
my bool $lck = !$lock.lockOwner();
if ($lck)
    $lock.lock();
on_exit
    if ($lck)
        $lock.unlock();
```

Returns

True if the calling thread owns the lock, False if not

45.6.2.4 int Qore::Thread::AbstractSmartLock::lockTID ( )

Returns the TID of the thread owning the lock or -1 if the lock is currently not acquired.

This method normally not useful in practice for anything except checking that the current thread owns the lock, in which case AbstractSmartLock::lockOwner() is better, because if the lock is not owned by the current thread the lock ownership can change at any time.

Code Flags:

CONSTANT

Example:

```
my int $tid = $lock.lockTID();
```

Returns

the TID of the thread owning the lock or -1 if the lock is currently not acquired

45.7 Qore::Thread::AutoGate Class Reference

A helper class for the Gate class for exception-safe Gate handling.

Public Member Functions

- constructor (Gate gate)

  Creates the AutoGate object based on the Gate argument passed and immediately calls Gate::enter()
- copy ()

  Throws an exception; objects of this class cannot be copied.
- destructor ()

  Calls Gate::exit() and destroys the AutoGate object.

45.7.1 Detailed Description

A helper class for the Gate class for exception-safe Gate handling.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

AutoGate objects, when used along with a Gate object, allow Qore programmers to safely enter and exit a Gate lock, even if exceptions are thrown or return statements are executed in the block where the AutoGate object is created.

AutoGate objects enter the gate lock for the lifetime of the AutoGate object. For this reason, it is only appropriate to assign an AutoGate object to a local variable, so when the local variable goes out of scope, the AutoGate object will be deleted and the gate automatically exited.

For example:

```
our Gate $gate();

sub check_error($error) {
    # note that the Gate is entered in the AutoGate constructor, and
    # the Gate will be exited as soon as the block is exited below.
    # (with either the throw statement or the return statement)
    my AutoGate $ag($gate);
    if ($error)
        throw "ERROR", "sorry, an error happened";

    return "OK";
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.7.2 Member Function Documentation

45.7.2.1 Qore::Thread::AutoGate::constructor (Gate *gate*)

Creates the AutoGate object based on the Gate argument passed and immediately calls Gate::enter()

Parameters

gate	the Gate object to enter for the lifetime of the AutoGate object

Example:

```
my AutoGate $ag($gate);
```

45.7.2.2 Qore::Thread::AutoGate::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

AUTOGATE-COPY-ERROR	objects of this class cannot be copied

45.7.2.3 Qore::Thread::AutoGate::destructor ( )

Calls Gate::exit() and destroys the AutoGate object.

Example:

```
delete $ag;
```

45.8 Qore::Thread::AutoLock Class Reference

A helper class for the Mutex class for exception-safe Mutex handling.

Public Member Functions

- constructor (Mutex mutex)

  Creates the AutoLock object based on the Mutex argument passed and immediately calls Mutex::lock()
- copy ()

  Throws an exception; objects of this class cannot be copied.
- destructor ()

  Calls Mutex::unlock() on the saved Mutex object and destroys the AutoLock object.
- nothing lock ()

  Attempts to relock the Mutex object being managed.
- int trylock ()

  Attempts to relock the Mutex object being managed; acquires the lock only if it is not already held; returns 0 for success (lock acquired) or -1 if the call would block.
- nothing unlock ()

  Unlocks the Mutex object being managed; wakes up one thread if any threads are blocked on this lock.

45.8.1 Detailed Description

A helper class for the Mutex class for exception-safe Mutex handling.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

AutoLock objects, when used along with a Mutex object, allow Qore programmers to safely acquire and release a Mutex lock, even if exceptions are thrown or return statements are executed in the block where the AutoLock object is created.

AutoLock objects are helper objects that acquire a Mutex for the lifetime of the object.

For this reason, it is only appropriate to assign an AutoLock object to a local variable, so when the local variable goes out of scope, the AutoLock object will be deleted and the Mutex will be automatically released.

For example:

```
our Mutex $mutex();

sub check_error($error) {
    # note that the Mutex is acquired in the AutoLock constructor, and
    # the Mutex will be released as soon as the block is exited below.
    # (with either the throw statement or the return statement)
    my AutoLock $al($mutex);
    if ($error)
        throw "ERROR", "sorry, an error happened";

    return "OK";
}
```

The destructor will call Mutex::unlock() only if the current thread owns the lock, so it is safe to unlock the lock manually (or by calling AutoLock::unlock()) while the AutoLock object is in scope.

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.8.2 Member Function Documentation

45.8.2.1 Qore::Thread::AutoLock::constructor (Mutex *mutex*)

Creates the AutoLock object based on the Mutex argument passed and immediately calls Mutex::lock()

The AutoLock object immediately calls Mutex::lock() on the Mutex object passed, and saves it; Mutex::unlock() is called in the destructor if the lock is still held by the current thread.

Example:

```
{
    # when the block exits, the lock is automatically released
    my AutoLock $al($mutex);
}
```

Parameters

mutex	a Mutex object to lock immediately and hold for the scope of the AutoLock object (unless manually unlocked)

Exceptions

LOCK-ERROR	lock called twice in the same thread, Mutex object has already been deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.8.2.2 Qore::Thread::AutoLock::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

AUTOLOCK-COPY-ERROR	Objects of this class cannot be copied

45.8.2.3 Qore::Thread::AutoLock::destructor ( )

Calls Mutex::unlock() on the saved Mutex object and destroys the AutoLock object.

Mutex::unlock() is only called if the current thread owns the lock

Example:

```
delete $al;
```

45.8.2.4 nothing Qore::Thread::AutoLock::lock ( )

Attempts to relock the Mutex object being managed.

Do not call this method unless the Mutex object being managed has been unlocked since the constructor

Example:

```
$al.lock();
```

Exceptions

LOCK-ERROR	lock called twice in the same thread, Mutex object has already been deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.8.2.5 int Qore::Thread::AutoLock::trylock ( )

Attempts to relock the Mutex object being managed; acquires the lock only if it is not already held; returns 0 for success (lock acquired) or -1 if the call would block.

Returns

0 for success (lock acquired) or -1 if the call would block (lock not acquired because it's held by another thread)

Example:

```
my int $i = $al.trylock();
```

Exceptions

LOCK-ERROR	object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.8.2.6 nothing Qore::Thread::AutoLock::unlock ( )

Unlocks the Mutex object being managed; wakes up one thread if any threads are blocked on this lock.

Example:

```
$al.unlock();
```

Exceptions

LOCK-ERROR	unlock called by a thread that does not own the lock or the lock is not locked, object deleted in another thread, etc

45.9 Qore::Thread::AutoReadLock Class Reference

A helper class for the RWLock class for exception-safe read lock handling.

Public Member Functions

- constructor (RWLock rwl)

  Creates the AutoReadLock object based on the RWLock argument passed and immediately calls RWLock::readLock()
- copy ()

  Throws an exception; objects of this class cannot be copied.
- destructor ()

  Calls RWLock::readUnlock() on the saved RWLock and destroys the AutoReadLock object.

45.9.1 Detailed Description

A helper class for the RWLock class for exception-safe read lock handling.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

AutoReadLock objects, when used along with a RWLock object, allow Qore programmers to safely acquire and release a read lock, even if exceptions are thrown or return statements are executed in the block where the AutoReadLock object is created.

AutoReadLock objects are helper objects that acquire a read lock for the lifetime of the AutoReadLock object. For this reason, it is only appropriate to assign an AutoReadLock object to a local variable, so when the local variable goes out of scope, the AutoReadLock object will be deleted and the read lock will be automatically released.

For example:

```
our RWLock $rwl();

sub check_error($error) {
    # note that the read lock is acquired in the AutoReadLock constructor, and
    # the read lock will be released as soon as the block is exited below.
    # (with either the throw statement or the return statement)
    my AutoReadLock $arl($rwl);
    if ($error)
        throw "ERROR", "sorry, an error happened";

    return "OK";
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option

45.9.2 Member Function Documentation

45.9.2.1 Qore::Thread::AutoReadLock::constructor (RWLock *rwl*)

Creates the AutoReadLock object based on the RWLock argument passed and immediately calls RWLock::readLock()

Creates the AutoReadLock object based on the RWLock argument passed. The AutoReadLock object immediately calls RWLock::readLock() on the RWLock object passed, and saves it so it can be released when the AutoReadLock object is destroyed.

Example:

```
my AutoReadLock $arl($rwlock);
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::readLock() called while already holding the write lock, object deleted in another thread, etc.

45.9.2.2 Qore::Thread::AutoReadLock::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

AUTOREADLOCK-COPY-ERROR	objects of this class cannot be copied

45.9.2.3 Qore::Thread::AutoReadLock::destructor ( )

Calls RWLock::readUnlock() on the saved RWLock and destroys the AutoReadLock object.

Example:

```
delete $arl;
```

Exceptions

LOCK-ERROR	RWLock::readUnlock() called while not holding the read lock, RWLock object deleted in another thread, etc

45.10 Qore::Thread::AutoWriteLock Class Reference

A helper class for the RWLock class for exception-safe write lock handling.

Public Member Functions

- constructor (RWLock rwl)

  Creates the AutoWriteLock object based on the RWLock argument passed and immediately calls RWLock::writeLock()
- copy ()

  Throws an exception; objects of this class cannot be copied.
- destructor ()

  Calls RWLock::writeUnlock() on the saved RWLock and destroys the AutoWriteLock object.

45.10.1 Detailed Description

A helper class for the RWLock class for exception-safe write lock handling.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

AutoWriteLock objects, when used along with a RWLock object, allow Qore programmers to safely acquire and release a write lock, even if exceptions are thrown or return statements are executed in the block where the AutoWriteLock object is created.

AutoWriteLock objects are helper objects that acquire a write lock for the lifetime of the AutoWriteLock object. For this reason, it is only appropriate to assign an AutoWriteLock object to a local variable, so when the local variable goes out of scope, the AutoWriteLock object will be deleted and the write lock will be automatically released.

For example:

```
our RWLock $rwl();

sub check_error($error) {
    # note that the write lock is acquired in the AutoWriteLock constructor, and
    # the write lock will be released as soon as the block is exited below.
    # (with either the throw statement or the return statement)
    my AutoWriteLock $awl($rwl);
    if ($error)
        throw "ERROR", "sorry, an error happened";

    return "OK";
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option

45.10.2 Member Function Documentation

45.10.2.1 Qore::Thread::AutoWriteLock::constructor (RWLock *rwl*)

Creates the AutoWriteLock object based on the RWLock argument passed and immediately calls RWLock::writeLock()

The AutoReadLock object immediately calls RWLock::writeLock() on the RWLock object passed, and saves it so it can be released when the AutoReadLock object is destroyed.

Example:

```
my AutoWriteLock $awl($rwlock);
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::writeLock() called while already holding the read lock, object deleted in another thread, etc.

45.10.2.2 Qore::Thread::AutoWriteLock::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

AUTOWRITELOCK-COPY-ERROR	objects of this class cannot be copied

45.10.2.3 Qore::Thread::AutoWriteLock::destructor ( )

Calls RWLock::writeUnlock() on the saved RWLock and destroys the AutoWriteLock object.

Example:

```
delete $awl;
```

Exceptions

LOCK-ERROR	RWLock::writeUnlock() called while not holding the write lock, RWLock object deleted in another thread, etc

45.11 Qore::Thread::Condition Class Reference

The Condition class can be used For blocking a thread until a condition becomes True.

Public Member Functions

- nothing broadcast ()

  Signals all threads blocked on this Condition object to wake up.
- constructor ()

  Creates the Condition object.
- copy ()

  Creates a new Condition object, not based on the original.
- nothing signal ()

  Signals a single blocked thread to wake up.
- int wait (AbstractSmartLock lock, timeout timeout_ms=0)

  Blocks a thread until signaled; accepts an optional timeout value.
- int wait_count (AbstractSmartLock lock)

  Returns the number of threads currently blocked on this object using the AbstractSmartLock passed.

45.11.1 Detailed Description

The Condition class can be used For blocking a thread until a condition becomes True.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

Condition objects, when used along with an AbstractSmartLock object (such as RWLock and Mutex objects), allow Qore threads to sleep until a certain condition becomes True.

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.11.2 Member Function Documentation

45.11.2.1 nothing Qore::Thread::Condition::broadcast ( )

Signals all threads blocked on this Condition object to wake up.

Normally this method call will be made while the same AbstractSmartLock object used for Condition::wait() calls is locked. Then, when the thread calling this method unlocks the AbstractSmartLock object, the thread(s) woken up by this call can continue executing.

Example:

```
$m.lock();
$cond.broadcast();
$m.unlock();
```

Exceptions

CONDITION-BROADCAST-ERROR	This exception should never be thrown - it indicates a low-level error in executing the method

45.11.2.2 Qore::Thread::Condition::constructor ( )

Creates the Condition object.

Example:

```
my Condition $cond();
```

45.11.2.3 Qore::Thread::Condition::copy ( )

Creates a new Condition object, not based on the original.

Example:

```
my Condition $new_cond = $cond.copy();
```

45.11.2.4 nothing Qore::Thread::Condition::signal ( )

Signals a single blocked thread to wake up.

Normally this method call will be made while the same AbstractSmartLock object used for Condition::wait() calls is locked. Then, when the thread calling this method unlocks the AbstractSmartLock object, the thread woken up by this call can continue executing.

Example:

```
$m.lock();
$cond.signal();
$m.unlock();
```

Exceptions

CONDITION-SIGNAL-ERROR	This exception should never be thrown - it indicates a low-level error in executing the method

45.11.2.5 int Qore::Thread::Condition::wait (AbstractSmartLock *lock*, timeout *timeout_ms* = 0)

Blocks a thread until signaled; accepts an optional timeout value.

Must be called with an AbstractSmartLock argument, and the AbstractSmartLock must be locked before the call. This method will atomically unlock the AbstractSmartLock object and wait on this Condition object to be woken up with a Condition::signal() or Condition::broadcast() method call in another thread. At this point, the AbstractSmartLock will be reacquired with the same state as it was acquired previously before control returns to the blocked thread. The wait condition should always be tested again when the thread is unblocked.

Parameters

lock	the AbstractSmartLock object to use for synchronization on this Condition object. The AbstractSmartLock must be locked before calling this method
timeout_ms	a timeout value to wait for the condition to be triggered; integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds). Timeout values <= 0 mean do not time out. If a timeout value > 0 is given and the call times out, the AbstractSmartLock will also be acquired when the Condition::wait() call returns and `ETIMEDOUT` will be returned.

Returns

0 for success, `ETIMEDOUT` if a timeout has occurred

Example:

```
$m.lock();
on_exit $m.unlock();
while ($some_value > 0) {
    $cond.wait($m);
}
printf("finally $some_value is 0\n");
```

Exceptions

CONDITION-WAIT-ERROR	This exception should never be thrown - it indicates a low-level error in executing the method

45.11.2.6 int Qore::Thread::Condition::wait_count (AbstractSmartLock *lock*)

Returns the number of threads currently blocked on this object using the AbstractSmartLock passed.

Parameters

lock	the AbstractSmartLock object to check for blocked threads on this Condition object; the AbstractSmartLock can be in any state (locked or unlocked) for this call (does not necessarily have to be locked).

Returns

The number of threads currently blocked on this object using the AbstractSmartLock object passed

Example:

```
printf("%d thread(s) waiting on the Condition\n", $cond.wait_count($m));
```

45.12 Qore::Thread::Counter Class Reference

Implements a class that can be used for blocking a thread until a counter reaches zero.

Public Member Functions

- constructor (softint c=0)

  Creates the Counter object.
- copy ()

  Creates a new Counter object with the same count as the original.
- nothing dec ()

  Atomically decrements the counter value.
- destructor ()

  Destroys the Counter object.
- int getCount ()

  Returns the current counter value.
- int getWaiting ()

  Returns the number of threads currently blocked on this object.
- nothing inc ()

  Atomically increments the counter value.
- nothing waitForZero ()

  Blocks a thread until the counter reaches zero.
- int waitForZero (timeout timeout_ms)

  Blocks a thread until the counter reaches zero.

45.12.1 Detailed Description

Implements a class that can be used for blocking a thread until a counter reaches zero.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

Counter objects allow Qore threads to sleep until a counter reaches zero.

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.12.2 Member Function Documentation

45.12.2.1 Qore::Thread::Counter::constructor (softint *c* = 0)

Creates the Counter object.

Parameters

c	an argument is supplied here, then the Counter will be initialized with this value, otherwise the Counter is initialized with 0

Example:

```
my Counter $counter();
```

Exceptions

COUNTER-ERROR	a negative number was passed to initialize the Counter

45.12.2.2 Qore::Thread::Counter::copy ( )

Creates a new Counter object with the same count as the original.

Example:

```
my Counter $new_counter = $counter.copy();
```

45.12.2.3 nothing Qore::Thread::Counter::dec ( )

Atomically decrements the counter value.

A `COUNTER-ERROR` exception can be thrown if the object is deleted in another thread while this call is in progress; this is a race condition caused by a user programming error and should not occur in practice with correct code.

Example:

```
$counter.dec();
```

Exceptions

COUNTER-ERROR	Counter has been deleted in another thread or Counter is already at 0

45.12.2.4 Qore::Thread::Counter::destructor ( )

Destroys the Counter object.

Note that it is a programming error to delete this object while other threads are blocked on it; in this case an exception is thrown in the deleting thread, and also in each thread blocked on this object when it is deleted.

Example:

```
delete $counter;
```

Exceptions

COUNTER-ERROR	Object deleted while other threads blocked on it

45.12.2.5 int Qore::Thread::Counter::getCount ( )

Returns the current counter value.

Returns

the current counter value

Code Flags:

CONSTANT

Example:

```
my int $c = $counter.getCount();
```

45.12.2.6 int Qore::Thread::Counter::getWaiting ( )

Returns the number of threads currently blocked on this object.

Returns

the number of threads currently blocked on this object

Code Flags:

CONSTANT

Example:

```
my int $c = $counter.getWaiting();
```

45.12.2.7 nothing Qore::Thread::Counter::inc ( )

Atomically increments the counter value.

Example:

```
$counter.inc();
```

45.12.2.8 nothing Qore::Thread::Counter::waitForZero ( )

Blocks a thread until the counter reaches zero.

Example:

```
$counter.waitForZero();
```

Exceptions

COUNTER-ERROR	Counter has been deleted in another thread

45.12.2.9 int Qore::Thread::Counter::waitForZero (timeout *timeout_ms*)

Blocks a thread until the counter reaches zero.

Parameters

timeout_ms	a timeout value to wait for the Counter to reach zero; integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds)

Returns

0 on sucess, or non-zero if a timeout occurred

Example:

```
if ($counter.waitForZero(1500))
    throw "TIMEOUT", "counter did not reach 0 in 1.5s";
```

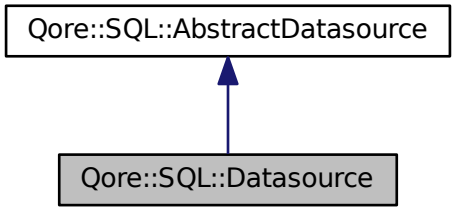
Exceptions

COUNTER-ERROR	Counter has been deleted in another thread

45.13 Qore::SQL::Datasource Class Reference

This class provides the Qore interface to databases.

Inheritance diagram for Qore::SQL::Datasource:



Public Member Functions

- nothing beginTransaction ()

  Manually grabs the transaction lock.
- nothing clearEventQueue ()

  Clears the queue object for DBI events on the datasource.
- nothing close ()

  Closes the connection to the database; if any actions are in progress on the database, the close call will block until the actions complete. If any errors are encountered, an exception is raised.
- nothing commit ()

  Commits the current transaction and releases the transaction lock.
- constructor (string driver, __7_ string user, __7_ string pass, __7_ string db, __7_ string encoding, __7_ string host, __7_ softint port, __7_ hash options, __7_ Qore::Thread::Queue queue, any arg)

  Creates the Datasource object; attempts to load a DBI driver if the driver is not already present in Qore.
- constructor (string desc, __7_ Qore::Thread::Queue queue, any arg)

  Creates a Datasource object from a single string giving all parameters that can be parsed by parse_datasource()
- constructor (hash opts, __7_ Qore::Thread::Queue queue, any arg)

  Creates a Datasource object from a hash argument giving parameters for the constructor.
- copy ()

  Creates a new Datasource object with the same driver as the original and copies of all the connection parameters.
- bool currentThreadInTransaction ()

  Returns True if the current thread is in a transaction (i.e. holds the transaction lock), False if not.
- __7_ hash describe (string sql,...)

  Executes an SQL select statement on the server and returns a description of the result set as a hash.
- destructor ()

  Closes the datasource if it's open (if any operations are in progress, will block until the operations complete) and destroys the object.
- any exec (string sql,...)

Grabs the transaction lock (if autocommit is disabled) and executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values).

- any execRaw (string sql)

  Grabs the transaction lock (if autocommit is disabled) and executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)

- bool getAutoCommit ()

  Returns the autocommit status for the object.

- int getCapabilities ()

  Returns an integer bitfield of DBI driver capabilities.

- list getCapabilityList ()

  Returns a list of strings giving the capabilities of the current DBI driver.

- any getClientVersion ()

  Retrieves the driver-specific client library version information; this method may not be implemented for all drivers.

- hash getConfigHash ()

  Returns a datasource hash describing the configuration of the current object.

- string getConfigString ()

  Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

- string getDBCharset ()

  Retrieves the database-specific charset set encoding for the current connection.

- string getDBEncoding ()

  Retrieves the database-specific charset set encoding for the current connection.

- __7_ string getDBName ()

  Returns the database name parameter as a string or NOTHING if none is set.

- string getDriverName ()

  Returns the name of the driver used for the object.

- __7_ string getHostName ()

  Returns the hostname parameter as a string or NOTHING if none is set.

- string getOSCharset ()

  Returns the Qore character encoding name for the current connection as a string or "(unknown)" if none is set.

- __7_ string getOSEncoding ()

  Returns the Qore character encoding name for the current connection as a string or NOTHING if none is set.

- any getOption (string opt)

  Returns the current value for the given option.

- hash getOptionHash ()

  returns the valid options for the driver associated with the Datasource with descriptions and current values for the current Datasource object

- __7_ string getPassword ()

  Returns the password parameter as a string or NOTHING if none is set.

- __7_ int getPort ()

  Gets the port number that will be used for the next connection to the server.

- any getServerVersion ()

  Returns the driver-specific server version data for the current connection.

- int getTransactionLockTimeout ()

  Retrieves the transaction lock timeout value as an integer in milliseconds.

- __7_ string getUserName ()

  Returns the username parameter as a string or NOTHING if none is set.

- bool inTransaction ()

  Returns True if a transaction is currently in progress, False if not.

- nothing open ()

Opens a connection to the datasouce, using the connection parameters already set; an exception is thrown if any errors occur.

- nothing reset ()

  Closes and reopens the Datasource.

- nothing rollback ()

  Rolls the current transaction back and releases the transaction lock.

- any select (string sql,...)

  Executes an SQL select statement on the server and returns the result as a hash (column names) of lists (column values per row)

- __7_ hash selectRow (string sql,...)

  Executes an SQL select statement on the server and returns the first row as a hash (the column values)

- any selectRows (string sql,...)

  Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

- nothing setAutoCommit (bool ac=True)

  Turns autocommit on or off for this object.

- nothing setDBCharset (string encoding)

  Sets the database-specific character encoding name for the next connection to the server.

- nothing setDBEncoding (string encoding)

  Sets the database-specific character encoding name for the next connection to the server.

- nothing setDBName (string db)

  Sets the database name parameter for the time a connection to the server is established.

- nothing setEventQueue (Qore::Thread::Queue queue, any arg)

  Sets a queue object for DBI events on the datasource.

- nothing setHostName (string host)

  Sets the hostname to use for the next connection to the server.

- setOption (string opt, any val)

  sets an option for the datasource

- nothing setPassword (string pass)

  Sets the password parameter for the time a connection to the server is established.

- nothing setPort (softint port=0)

  Sets the port number to use for the connection.

- nothing setTransactionLockTimeout (timeout timeout_ms=0)

  Sets the transaction lock timeout value in milliseconds; set to 0 for no timeout.

- nothing setUserName (string user)

  Sets the username parameter for the time a connection to the server is established.

- int transactionTid ()

  Returns the TID of the thread holding the transaction lock or -1 if it's not currently held.

- any vexec (string sql, __7_ softlist vargs)

  Grabs the transaction lock (if autocommit is disabled) and executes SQL code on the DB connection, taking a list for all bind arguments.

- any vselect (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.

- __7_ hash vselectRow (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.

- any vselectRows (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

45.13.1 Detailed Description

This class provides the Qore interface to databases.

Restrictions:

Qore::PO_NO_DATABASE

This class provides the main direct interface to DBI drivers (along with the SQLStatement and DatasourcePool classes).

The Datasource class will attempt to load any DBI driver that is not currently loaded in the constructor. For connection pooling support, see the DatasourcePool class.

Datasource objects will implicitly call Datasource::open() if no connection has yet been established and a method is called requiring a connection to the database server. Therefore any method that requires communication with the database server can also throw any exception that the open method can throw.

Most Qore DBI drivers allow "select" queries to be executed through the Datasource::exec() method, and allow SQL commands (procedure calls, etc) to be executed through the Datasource::select() method, and some DBI drivers do not (depends on the underlying DB API). At any rate, the transaction lock is set when auto-commit is disabled and when the Datasource::exec() or Datasource::beginTransaction() methods are executed as documented above. Therefore executing a transaction relevant command through the Datasource::select() method while auto-commit mode is disabled and a transaction has not yet started will not result in the transaction lock being allocated to the current thread and therefore could cause transaction errors when sharing the Datasource object between multiple threads.

Only databases with an existing Qore DBI driver can be accessed through the Datasource class.

All Qore DBI drivers set new connections to use transaction isolation level `"read committed"`.

The Datasource class provides high-level, per-connection locking on requests at a level above the DBI drivers to ensure that the communication between clients and servers is properly serialized.

Datasource objects also have a default character encoding; all requests to the server will be made in this encoding, and all responses will be returned in the given encoding. If no encoding is specifically given to the Datasource object, the Datasource object will use the Default Character Encoding.

Datasource Binding By Value and By Placeholder

All Datasource methods accepting SQL strings to execute understand a special syntax used in the query string to bind Qore data by value and to specify placeholders for output variables (for example, when executing a stored procedure or database function). Placeholder binding is DBI driver specific, but binding by value is supported with the same syntax in all drivers. Additionally, the `%d` numeric specifier is supported equally in all Qore DBI drivers.

Datasource Format Specifiers

Format Specification	Description
`%d`	If any value other than NOTHING or NULL is given, then the value is converted to an integer and this value is substituted in the string at this position; if the value is NOTHING or NULL, then a literal `"null"` is substituted instead.
`%s`	The argument is converted to a string and the string is inserted literally without any conversion or escape sequences in the string; this is useful for table or schema prefixes, etc

%v	The argument is bound by value according to the DBI driver's implementation.

To bind Qore data values directly in a binary format in an SQL command, use %v in the command string, and include the value as an argument after the string. Binding by value allows the DBI driver to take care of formatting the data properly for use in the query with the database server. When binding by value, strings do not need to be quoted, date/time values do not need special formatting, binary objects (with BLOB columns, for example) can be used directly in queries, etc.

Here is an example:

```
my int $rows = $db.exec("insert into table (varchar_col, timestamp_col, blob_col, numeric_col) values (%v,
       %v, %v, %d)", $string, now(), $binary, 100);
```

When using dynamic SQL, to insert a numeric value or a literal "null" in a query, use %d in the command string, and include the value as an argument after the string. If the value is NOTHING or NULL, a literal "null" will be written to the string; otherwise the argument is converted to an arbitrary-precision number or integer value if necessary and written to the string. This is useful for working with DECIMAL (NUMERIC, NUMBER) types in a database-independent way; for example PostgreSQL servers do not do type conversions to DECIMAL types when a string, integer, or float is bound by value, therefore to ensure that integral decimal values can be used in a database-independent way (with "null" substitution when no value is bound), a valid approach is to use the %d code in the command string instead of %v. However please note that %v is normally preferred to keep the server-side statement cache a manageable size, in the previous example using the int(), float(), or number() functions to convert string values before binding with %v may be better.

For binding placeholders for output variables, write a unique name in the string and prefix it with a colon (ex: "↩ :code"). In this case the method will return a hash of the output variables using the placeholder names as keys, but without the colon prefix. By default, a string type will be bound to the position. To bind other variable types to placeholder positions, include the type constant (see SQL Constants) as an argument after the command string. For BLOBs, use Binary, for CLOBs, use the string "clob" (constants will be provided in a future release). Not all DBI drivers require placeholder buffer specifications; see the documentation for the DBI driver in question for more information and examples regarding placeholder buffer specifications.

Datasource Transaction Locks

Datasource objects have an internal transaction lock which will be grabbed when the Datasource::exec(), Datasource::vexec() Datasource::execRaw(), or Datasource::beginTransaction() methods are executed and autocommit is not enabled. This enables a single datasource to be safely used for transaction management by several threads simultaneously. Note that an exception in a Datasource method that would acquire the lock (such as the Datasource::exec() method) when it's not already held, will have the effect that the transaction lock is not acquired.

Any thread attempting to do transaction-relevant actions on a Datasource with auto-commit disabled while a transaction is in progress by another thread will block until the thread currently executing a transaction executes the Datasource::commit() or Datasource::rollback() methods (or the Datasource is deleted, reset, or closed, in which case the lock is released and an exception is raised as well).

There is a timeout associated with the transaction lock; if a thread waits for the transaction lock for more than the timeout period, then an exception will be raised in the waiting thread. The timeout value can be read and changed with the Datasource::getTransactionLockTimeout() and Datasource::setTransactionLockTimeout() methods, respectively. The default transaction lock timeout value is 120 seconds.

Note that the SQLStatement class also grabs the transaction lock when executing if it is created using a Datasource object in the constructor; for more information see the SQLStatement class.

Note

This class is not available with the PO_NO_DATABASE parse option

See also

SqlUtil for a high level database-independent API

45.13.2 Member Function Documentation

45.13.2.1 nothing Qore::SQL::Datasource::beginTransaction ( ) [virtual]

Manually grabs the transaction lock.

This method should be called when the Datasource object will be shared between more than 1 thread, and a transaction will be started with a Datasource::select() method or the like.

This method does not make any communication with the server to start a transaction; it only allocates the transaction lock to the current thread in Qore.

It is an error to call this method when autocommit is enabled for the Datasource.

Example:

```
$db.beginTransaction();
```

Exceptions

AUTOCOMMIT-ERROR	Cannot start a transaction when autocommit is enabled
TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Implements Qore::SQL::AbstractDatasource.

45.13.2.2 nothing Qore::SQL::Datasource::clearEventQueue ( )

Clears the queue object for DBI events on the datasource.

Since

Qore 0.8.9

45.13.2.3 nothing Qore::SQL::Datasource::close ( )

Closes the connection to the database; if any actions are in progress on the database, the close call will block until the actions complete. If any errors are encountered, an exception is raised.

Example:

```
$db.close();
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

45.13.2.4 nothing Qore::SQL::Datasource::commit ( ) [virtual]

Commits the current transaction and releases the transaction lock.

Example:

```
$db.commit();
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.5 Qore::SQL::Datasource::constructor (string *driver*, __7_ string *user*, __7_ string *pass*, __7_ string *db*, __7_ string *encoding*, __7_ string *host*, __7_ softint *port*, __7_ hash *options*, __7_ Qore::Thread::Queue *queue*, any *arg*)

Creates the Datasource object; attempts to load a DBI driver if the driver is not already present in Qore.

Parameters

driver	The name of the DBI driver for the Datasource. See SQL Constants for builtin constants for DBI drivers shipped with Qore, or see the DBI driver documentation to use an add-on driver (this string should be the name of the driver to be loaded)
user	The user name for the new connection. Also see Datasource::setUserName() for a method that allows this parameter to be set after the constructor.
pass	The password for the new connection. Also see Datasource::setPassword() for a method that allows this parameter to be set after the constructor.
db	The database name for the new connection. Also see Datasource::setDBName() for a method that allows this parameter to be set after the constructor.
encoding	The database-specific name of the character encoding to use for the new connection. Also see Datasource::setDBCharset() for a method that allows this parameter to be set after the constructor. If no value is passed for this parameter, then the database character encoding corresponding to the default character encoding will be used instead.
host	The host name for the new connection. Also see Datasource::setHostName() for a method that allows this parameter to be set after the constructor.
port	The port number for the new connection. Also see Datasource::setPort() for a method that allows this parameter to be set after the constructor.
options	Any options for the database driver for the new connection
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a `QUEUE-ERROR` will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the `arg` key of datasource events

Example:

```
my Datasource $db(DSPGSQL, "user", "pass", "database", "utf8", "localhost", 5432);
```

Exceptions

DATASOURCE-UNSUPPORTED-DATABASE	DBI driver cannot be loaded
DATASOURCE-CONSTRUCTOR-ERROR	port value is <= 0
DBI-OPTION-ERROR	unknown or unsupported option passed to driver

45.13.2.6 Qore::SQL::Datasource::constructor (string *desc*, __7_ Qore::Thread::Queue *queue*, any *arg*)

Creates a Datasource object from a single string giving all parameters that can be parsed by parse_datasource()

Parameters

desc	a datasource description string in the format that can be parsed by parse_datasource()
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a `QUEUE-ERROR` will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the `arg` key of datasource events

Example:

```
my Datasource ds("pgsql:user/pass@db01(utf8)%localhost:5432");
```

Exceptions

DATASOURCE-UNSUPP↩ ORTED-DATABASE	DBI driver cannot be loaded
DATASOURCE-CONSTR↩ UCTOR-ERRO	missing required parameter for connection; port value is <= 0
DBI-OPTION-ERROR	unknown or unsupported option passed to driver

Since

Qore 0.8.6

45.13.2.7 Qore::SQL::Datasource::constructor (hash *opts*, _7_ Qore::Thread::Queue *queue*, any *arg*)

Creates a Datasource object from a hash argument giving parameters for the constructor.

Parameters

opts	a hash giving parameters for the new datasource with the following possible keys (the "type" key is mandatory, also usable with the output of the parse_datasource() function): • type: (*string) The name of the database driver to use; this key is mandatory; if not present, an exception will be raised. See SQL Constants for builtin constants for DBI drivers shipped with Qore, or see the DBI driver documentation to use an add-on driver (this string should be the name of the driver to be loaded) • user: (*string) The user name for the new connection. Also see Datasource::setUserName() for a method that allows this parameter to be set after the constructor. • pass: (*string) The password for the new connection. Also see Datasource::setPassword() for a method that allows this parameter to be set after the constructor. • db: (*string) The database name for the new connection. Also see Datasource::setDBName() for a method that allows this parameter to be set after the constructor. • charset: (*string) The database-specific name of the character encoding to use for the new connection. Also see Datasource::setDBCharset() for a method that allows this parameter to be set after the constructor. If no value is passed for this parameter, then the database character encoding corresponding to the default character encoding for the Qore process will be used instead. • host: (*string) The host name for the new connection. Also see Datasource::setHostName() for a method that allows this parameter to be set after the constructor. • port: (softint) The port number for the new connection. Also see Datasource::setPort() for a method that allows this parameter to be set after the constructor. If this key is present and is 0 then an exception will be raised. • options: (*hash) Any options for the new connection
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the arg key of datasource events

Example:

```
my Datasource $db(("type": DSPGSQL, "user": "username", "pass": "password", "db": "database", "
      charset": "utf8", "host": "localhost", "port": 5432);
```

Exceptions

DATASOURCE-UNSUPPORTED-DATABASE	DBI driver cannot be loaded
DATASOURCE-CONSTRUCTOR-ERROR	missing or invalid "driver" key, other key name not assigned to a string; port value is <= 0
DBI-OPTION-ERROR	unknown or unsupported option passed to driver

45.13.2.8 Qore::SQL::Datasource::copy ( )

Creates a new Datasource object with the same driver as the original and copies of all the connection parameters.

Example:

```
my Datasource $new_ds = $ds.copy();
```

45.13.2.9 bool Qore::SQL::Datasource::currentThreadInTransaction ( )

Returns True if the current thread is in a transaction (i.e. holds the transaction lock), False if not.

Returns

True if the current thread is in a transaction (i.e. holds the transaction lock), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $db.currentThreadInTransaction();
```

See also

Datasource::transactionTid()

Since

Qore 0.8.7

45.13.2.10 __7_ hash Qore::SQL::Datasource::describe (string *sql*, ...)

Executes an SQL select statement on the server and returns a description of the result set as a hash.

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using `:`*key_name* in the command string) in order after the command string

Returns

a hash describing the result set with the following keys:

- `name`: the name of the column
- `type`: a string giving the type of the column
- `maxsize`: an integer giving the column size
- `native_type`: a string giving the native column type
- `internal_id`: an integer giving the native column type code

Example:

```
my *hash $h = $db.describe("select * from example_table where id = 1");
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Since

Qore 0.8.9

45.13.2.11 Qore::SQL::Datasource::destructor ( )

Closes the datasource if it's open (if any operations are in progress, will block until the operations complete) and destroys the object.

Example:

```
delete $db;
```

Exceptions

DATASOURCE-TRANSACTION-EXCEPTION	The Datasource was destroyed while a transaction was still in progress; the transaction will be automatically rolled back

45.13.2.12 any Qore::SQL::Datasource::exec (**string** *sql*, *...*) [virtual]

Grabs the transaction lock (if autocommit is disabled) and executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values).

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.exec("insert into table (varchar_col, timestamp_col, blob_col, numeric_col) values (%v,
       %v, %v, %d)", $string, now(), $binary, 100);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.13 any Qore::SQL::Datasource::execRaw (string *sql*) [virtual]

Grabs the transaction lock (if autocommit is disabled) and executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)

This method does not do any variable binding, so it's useful for example for DDL statements etc

Warning:

Using this method to execute pure dynamic SQL many times with different SQL strings (as opposed to using the same string and binding by value instead of dynamic SQL) can affect application performance by prohibiting the efficient usage of the DB server's statement cache. See DB server documentation for variable binding and the SQL statement cache for more information.

Parameters

sql	The SQL command to execute on the server; this string will not be subjected to any transformations for variable binding

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
$db.execRaw("create table my_tab (id number, some_text varchar2(30))");
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.14 bool Qore::SQL::Datasource::getAutoCommit ( )

Returns the autocommit status for the object.

Returns

the autocommit status for the object

Code Flags:

CONSTANT

Example:

```
my bool $b = $db.getAutoCommit();
```

45.13.2.15 int Qore::SQL::Datasource::getCapabilities ( )

Returns an integer bitfield of DBI driver capabilities.

Returns

an integer bitfield of DBI driver capabilities; see DBI Capability Constants for the meaning of each bit

Code Flags:

CONSTANT

Example:

```
my int $caps = $db.getCapabilities();
if (!($caps & DBI_CAP_TRANSACTION_MANAGEMENT))
    throw "DATASOURCE-ERROR", sprintf("DBI driver %y does not support transaction management", $db.
      getDriverName());
```

45.13.2.16 list Qore::SQL::Datasource::getCapabilityList ( )

Returns a list of strings giving the capabilities of the current DBI driver.

Returns

a list of strings giving the capabilities of the current DBI driver

Code Flags:

CONSTANT

Example:

```
printf("driver %y has the following capabilities:\n", $db.getDriverName());
foreach my string $cap in ($db.getCapabilityList())
    printf("- %s\n", $cap);
```

45.13.2.17 any Qore::SQL::Datasource::getClientVersion ( ) [virtual]

Retrieves the driver-specific client library version information; this method may not be implemented for all drivers.

Returns

the driver-specific client library version information; this method may not be implemented for all drivers; see the DBI driver documentation for the return data type and format

Example:

```
my any $ver = $db.getClientVersion();
```

Note

see the documentation for the DBI driver being used for possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.18 hash Qore::SQL::Datasource::getConfigHash ( ) [virtual]

Returns a datasource hash describing the configuration of the current object.

Code Flags:

CONSTANT

Example:

```
my hash $h = $ds.getConfigHash();
```

Returns

a datasource hash describing the configuration of the current object

Since

Qore 0.8.8

Implements Qore::SQL::AbstractDatasource.

45.13.2.19 string Qore::SQL::Datasource::getConfigString ( ) [virtual]

Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Code Flags:

CONSTANT

Example:

```
my string $str = $ds.getConfigString();
```

Returns

a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Since

Qore 0.8.8

Implements Qore::SQL::AbstractDatasource.

45.13.2.20 string Qore::SQL::Datasource::getDBCharset ( )

Retrieves the database-specific charset set encoding for the current connection.

A method synonym for Datasource::getDBEncoding() kept for backwards-compatibility.

Returns

the database-specific charset set encoding for the current connection

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getDBCharset();
```

45.13.2.21 string Qore::SQL::Datasource::getDBEncoding ( ) [virtual]

Retrieves the database-specific charset set encoding for the current connection.

Returns

the database-specific charset set encoding for the current connection

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getDBEncoding();
```

See also

Datasource::getOSEncoding();

Implements Qore::SQL::AbstractDatasource.

45.13.2.22 __7_ string Qore::SQL::Datasource::getDBName ( ) [virtual]

Returns the database name parameter as a string or NOTHING if none is set.

Returns

the database name parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $db = $db.getDBName();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.23 string Qore::SQL::Datasource::getDriverName ( ) [virtual]

Returns the name of the driver used for the object.

Returns

the name of the driver used for the object

Code Flags:

CONSTANT

Example:

```
my string $driver = $db.getDriverName();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.24 __7_ string Qore::SQL::Datasource::getHostName ( ) [virtual]

Returns the hostname parameter as a string or NOTHING if none is set.

Returns

the hostname parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $host = $db.getHostName();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.25 any Qore::SQL::Datasource::getOption (string *opt*)

Returns the current value for the given option.

Code Flags:

RET_VALUE_ONLY

Parameters

opt	the option to get

Note

in order to ensure atomicity when dealing with Datasource options, the transaction lock is acquired before executing this method if it was not already owned by the calling thread

Exceptions

DBI-OPTION-ERROR	unknown or unsupported option passed to driver
TRANSACTION-LOCK-TI↩ MEOUT	Timeout trying to acquire the transaction lock

Since

Qore 0.8.6

45.13.2.26 hash Qore::SQL::Datasource::getOptionHash ( )

returns the valid options for the driver associated with the Datasource with descriptions and current values for the current Datasource object

Code Flags:

RET_VALUE_ONLY

Returns

a hash where the keys are valid option names, and the values are hashes with the following keys:

- `"desc"`: a string description of the option
- `"type"`: a string giving the data type restriction for the option
- `"value"`: the current value of the option

Note

in order to ensure atomicity when dealing with Datasource options, the transaction lock is acquired before executing this method if it was not already owned by the calling thread

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Since

Qore 0.8.6

45.13.2.27 string Qore::SQL::Datasource::getOSCharset ( )

Returns the Qore character encoding name for the current connection as a string or `"(unknown)"` if none is set.

Returns

the Qore character encoding name for the current connection as a string or `"(unknown)"` if none is set

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getOSCharset();
```

See also

Datasource::getOSEncoding()

45.13.2.28 __7_ string Qore::SQL::Datasource::getOSEncoding ( ) [virtual]

Returns the Qore character encoding name for the current connection as a string or NOTHING if none is set.

Returns

the Qore character encoding for the current connection as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $enc = $db.getOSEncoding();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.29 __7_ string Qore::SQL::Datasource::getPassword ( ) [virtual]

Returns the password parameter as a string or NOTHING if none is set.

Returns

the password parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $pass = $db.getPassword();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.30 __7_ int Qore::SQL::Datasource::getPort ( ) [virtual]

Gets the port number that will be used for the next connection to the server.

Invalid port numbers will cause an exception to be thrown when the connection is opened

Code Flags:

CONSTANT

Example:

```
my *int $port = $db.getPort();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.31 any Qore::SQL::Datasource::getServerVersion ( ) [virtual]

Returns the driver-specific server version data for the current connection.

Returns

the driver-specific server version data for the current connection; see the DBI driver documentation for the return data type and format

Example:

```
my any $ver = $db.getServerVersion();
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.32 int Qore::SQL::Datasource::getTransactionLockTimeout ( )

Retrieves the transaction lock timeout value as an integer in milliseconds.

Returns

the transaction lock timeout value as an integer in milliseconds

Code Flags:

CONSTANT

Example:

```
my int $to_ms = $db.getTransactionLockTimeout();
```

45.13.2.33 __7_ string Qore::SQL::Datasource::getUserName ( ) [virtual]

Returns the username parameter as a string or NOTHING if none is set.

Returns

the username parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $user = $db.getUserName();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.34 bool Qore::SQL::Datasource::inTransaction ( ) [virtual]

Returns True if a transaction is currently in progress, False if not.

Returns

True if a transaction is currently in progress, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $db.inTransaction();
```

Implements Qore::SQL::AbstractDatasource.

45.13.2.35 nothing Qore::SQL::Datasource::open ( )

Opens a connection to the datasouce, using the connection parameters already set; an exception is thrown if any errors occur.

If the connection is already open, then no action is taken.

Example:

```
$db.open();
```

Note

see the documentation for the DBI driver being used for possible exceptions

See also

Datasource::reset()

45.13.2.36 nothing Qore::SQL::Datasource::reset ( )

Closes and reopens the Datasource.

Example:

```
$db.reset();
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

45.13.2.37 nothing Qore::SQL::Datasource::rollback ( ) [virtual]

Rolls the current transaction back and releases the transaction lock.

Example:

```
$db.rollback();
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.38 any Qore::SQL::Datasource::select (string *sql*, ...) [virtual]

Executes an SQL select statement on the server and returns the result as a hash (column names) of lists (column values per row)

The return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

Additionally, this format is a more efficient format than that returned by the Datasource::selectRows() method, because the column names are not repeated for each row returned. Therefore, for retrieving anything greater than small amounts of data, it is recommended to use this method instead of Datasource::selectRows().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method returns a hash (the keys are the column names) of lists (the column data per row) when executed with an SQL select statement, however some DBI drivers allow any SQL to be executed through this method, in which case other data types can be returned (such as an integer for a row count or a hash for output parameters when executing a stored procedure)

Example:

```
# bind a string and a date/time value by value in a query
$query = $db.select("select * from table where varchar_column = %v and timestamp_column > %v", $string,
       2007-10-11T15:31:26.289);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

- Datasource::vselect()
- Qore::zzz8hashzzz9::contextIterator()
- context Statements

Implements Qore::SQL::AbstractDatasource.

45.13.2.39 __7_ hash Qore::SQL::Datasource::selectRow (string *sql*, ...) [virtual]

Executes an SQL select statement on the server and returns the first row as a hash (the column values)

If more than one row is returned, then it is treated as an error and a `DBI-SELECT-ROW-ERROR` is returned (however the DBI driver should raise its own exception here to avoid retrieving more than one row from the server). For a similar method taking a list for all bind arguments, see Datasource::vselectRow().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement

Example:

```
my *hash $h = $db.selectRow("select * from example_table where id = 1");
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock
DBI-SELECT-ROW-ERROR	more than 1 row retrieved from the server

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.40 any Qore::SQL::Datasource::selectRows (**string** *sql*, ...) [virtual]

Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

The return format of this method is not as memory efficient as that returned by the Datasource::select() method, therefore for larger amounts of data, it is recommended to use Datasource::select().

The usual return value of this method can be iterated with the ListHashIterator class.

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.selectRows("select * from example_table");
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

Datasource::select()

Implements Qore::SQL::AbstractDatasource.

45.13.2.41 nothing Qore::SQL::Datasource::setAutoCommit (bool *ac* = True)

Turns autocommit on or off for this object.

Parameters

ac	True to turn on autocommit (a commit will be executed after every SQL call to the server), False to turn off autocommit (commits must be manually triggered)

Example:

```
$db.setAutoCommit(False);
```

45.13.2.42 nothing Qore::SQL::Datasource::setDBCharset (string *encoding*)

Sets the database-specific character encoding name for the next connection to the server.

This is a method synonym for the Datasource::setDBEncoding() method, kept for backwards-compatibility.

Invalid character encoding names will cause an exception to be thrown when the connection is opened.

Parameters

encoding	the database-specific character encoding name for the next connection to the server

Example:

```
$db.setDBCharset($encoding);
```

45.13.2.43 nothing Qore::SQL::Datasource::setDBEncoding (string *encoding*)

Sets the database-specific character encoding name for the next connection to the server.

Invalid character encoding names will cause an exception to be thrown when the connection is opened.

Parameters

encoding	the database-specific character encoding name for the next connection to the server

Example:

```
$db.setDBEncoding($encoding);
```

45.13.2.44 nothing Qore::SQL::Datasource::setDBName (string *db*)

Sets the database name parameter for the time a connection to the server is established.

Invalid database names will cause an exception to be thrown when the connection is opened

Parameters

db	the database name parameter for the time a connection to the server is established

Example:

```
$db.setDBName($db);
```

45.13.2.45 nothing Qore::SQL::Datasource::setEventQueue (Qore::Thread::Queue *queue*, any *arg*)

Sets a queue object for DBI events on the datasource.

Parameters

queue	the Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a `QUEUE-ERROR` will be thrown; passing NOTHING will clear any event queue
arg	an argument to be included in the `arg` key of datasource events

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

Since

Qore 0.8.9

45.13.2.46 nothing Qore::SQL::Datasource::setHostName (string *host*)

Sets the hostname to use for the next connection to the server.

Invalid hostnames will cause an exception to be thrown when the connection is opened.

Parameters

host	the hostname to use for the next connection to the server

Example:

```
$db.setHostName($host);
```

45.13.2.47 Qore::SQL::Datasource::setOption (string *opt*, any *val*)

sets an option for the datasource

Parameters

opt	the option to set
val	the value to set

Note

in order to ensure atomicity when dealing with Datasource options, the transaction lock is acquired before executing this method if it was not already owned by the calling thread

Exceptions

DBI-OPTION-ERROR	unknown or unsupported option passed to driver
TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Since

Qore 0.8.6

45.13.2.48 nothing Qore::SQL::Datasource::setPassword (string *pass*)

Sets the password parameter for the time a connection to the server is established.

Invalid passwords will cause an exception to be thrown when the connection is opened

Parameters

pass	the password parameter for the time a connection to the server is established

Example:

```
$db.setPassword($pass);
```

45.13.2.49 nothing Qore::SQL::Datasource::setPort (softint *port* = 0)

Sets the port number to use for the connection.

Invalid port numbers will cause an exception to be thrown when the connection is opened

Parameters

port	the port number to use for the connection

Example:

```
$db.setPort($port);
```

45.13.2.50 nothing Qore::SQL::Datasource::setTransactionLockTimeout (timeout *timeout_ms* = 0)

Sets the transaction lock timeout value in milliseconds; set to 0 for no timeout.

Parameters

timeout_ms	the transaction lock timeout value in milliseconds; set to 0 for no timeout. Like all Qore functions and methods taking timeout values, a relative date/time value may be passed instead of an integer to make the timeout units clear (ex: `2500ms` for 2.5 seconds).

Example:

```
$db.setTransactionLockTimeout(4s);
```

45.13.2.51 nothing Qore::SQL::Datasource::setUserName (string *user*)

Sets the username parameter for the time a connection to the server is established.

Invalid usernames will cause an exception to be thrown when the connection is opened

Parameters

user	the username parameter for the time a connection to the server is established

Example:

```
$db.setUserName($user);
```

45.13.2.52 int Qore::SQL::Datasource::transactionTid ( )

Returns the TID of the thread holding the transaction lock or -1 if it's not currently held.

If the Datasource object is used in a multithreaded context and if the transaction lock is not held by the current thread, then the transaction lock status could change at any time.

Returns

the TID of the thread holding the transaction lock or -1 if it's not currently held

Code Flags:

CONSTANT

Example:

```
my int $tid = $db.transactionTid();
```

See also

Datasource::currentThreadInTransaction()

Since

Qore 0.8.7

45.13.2.53 any Qore::SQL::Datasource::vexec (string *sql*, __7_ softlist *vargs*) [virtual]

Grabs the transaction lock (if autocommit is disabled) and executes SQL code on the DB connection, taking a list for all bind arguments.

Same as Datasource::exec() except takes an explicit list for bind arguments

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.vexec("insert into example_table value (%v, %v, %v)", $arg_list);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.13.2.54 any Qore::SQL::Datasource::vselect (**string** *sql*, __7_ **softlist** *vargs*) [virtual]

Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.

The return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a hash (the keys are the column names) of list (each hash key's value is a list giving the row data), however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $query = $db.vselect("select * from example_table where id = %v and name = %v", $arg_list);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

- Datasource::select()
- Qore::zzz8hashzzz9::contextIterator()
- context Statements

Implements Qore::SQL::AbstractDatasource.

45.13.2.55 __7_ hash Qore::SQL::Datasource::vselectRow (string *sql*, __7_ softlist *vargs*) [virtual]

Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.

This method is the same as the Datasource::selectRow() method, except this method takes a single argument after the SQL command giving the list of bind value parameters

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement

Example:

```
my *hash $h = $db.vselectRow("select * from example_table where id = %v and type = %v", $arg_list);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

see the documentation for the DBI driver being used for additional possible exceptions

See also

Datasource::selectRow()

Implements Qore::SQL::AbstractDatasource.

45.13.2.56 any Qore::SQL::Datasource::vselectRows (string *sql*, _7_ softlist *vargs*) [virtual]

Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

Same as the Datasource::selectRows() method, except this method takes a single argument after the SQL command giving the list of bind value parameters.

The usual return value of this method can be iterated with the ListHashIterator class.

The return format of this method is not as memory efficient as that returned by the Datasource::select() method, therefore for larger amounts of data, it is recommended to use Datasource::select().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the transaction lock if it was not already acquired before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute Datasource::beginTransaction() first to ensure that the transaction lock is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.vselectRows("select * from example_table where id = %v and type = %v", $arg_list);
```

Exceptions

TRANSACTION-LOCK-TIMEOUT	Timeout trying to acquire the transaction lock

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

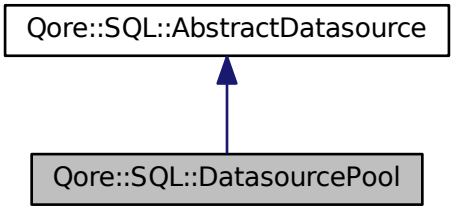
Datasource::selectRows()

Implements Qore::SQL::AbstractDatasource.

45.14 Qore::SQL::DatasourcePool Class Reference

Provides transparent per-thread, per-transaction datasource connection pooling.

Inheritance diagram for Qore::SQL::DatasourcePool:



Public Member Functions

- nothing beginTransaction ()

  Manually allocates a persistent connection from the pool to the calling thread.
- nothing clearEventQueue ()

  Clears the queue object for DBI events on the pool.
- clearWarningCallback ()

  clears any connection delay warning callback from the object
- nothing commit ()

  Commits the current transaction and releases the connection to the pool.
- constructor (string driver, __7_ string user, __7_ string pass, __7_ string db, __7_ string encoding, __7_ string host, softint min=3, softint max=10, softint port=0, __7_ Qore::Thread::Queue queue, any arg)

  Creates the DatasourcePool object; attempts to load a DBI driver if the driver is not already present in Qore.
- constructor (string desc, __7_ Qore::Thread::Queue queue, any arg)

  Creates a DatasourcePool object from a single string giving all parameters that can be parsed by parse_datasource()
- constructor (hash opts, __7_ Qore::Thread::Queue queue, any arg)

  Creates a Datasource object from a hash argument giving parameters for the constructor.
- copy ()

  Creates a new Datasource object with the same driver as the original and copies of all the connection parameters.
- bool currentThreadInTransaction ()

  Returns True if the current thread is in a transaction (i.e. has a dedicated datasource allocation), False if not.
- destructor ()

  Throws an exception if any transactions are in progress and returns immediately; the object is destroyed after any in-progress requests are completed.
- any exec (string sql,...)

  Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values)
- any execRaw (string sql)

  Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)
- any getClientVersion ()

  Retrieves the driver-specific client library version information; this method may not be implemented for all drivers.
- hash getConfigHash ()

  Returns a datasource hash describing the configuration of the current object.
- string getConfigString ()

Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

- __7_ string getDBCharset ()

  Retrieves the database-specific charset set encoding for the object.

- string getDBEncoding ()

  Retrieves the database-specific charset set encoding for the object.

- __7_ string getDBName ()

  Returns the database name parameter as a string or NOTHING if none is set.

- string getDriverName ()

  Returns the name of the driver used for the object.

- int getErrorTimeout ()

  Returns the error timeout period for waiting for a connection to come free as an integer giving milliseconds; note that timeout values less than or equal to zero mean that requests for a connection will wait indefinitely.

- __7_ string getHostName ()

  Returns the hostname parameter as a string or NOTHING if none is set.

- int getMaximum ()

  Returns the maximum number of connections in this object.

- int getMinimum ()

  Returns the minimum number of connections in this object.

- string getOSCharset ()

  Returns the Qore character encoding name for the object as a string or "(unknown)" if none is set.

- __7_ string getOSEncoding ()

  Returns the Qore character encoding name for the object as a string or NOTHING if none is set.

- any getOption (string opt)

  Returns the current value for the given option.

- hash getOptionHash ()

  returns the valid options for the driver associated with the Datasource with descriptions and current values for the current Datasource object

- __7_ string getPassword ()

  Returns the password parameter as a string or NOTHING if none is set.

- __7_ int getPort ()

  Gets the port number that will be used for the next connection to the server.

- any getServerVersion ()

  Returns the driver-specific server version data for the current connection.

- __7_ hash getUsageInfo ()

  Returns a hash with usage information about the DatasourcePool object.

- __7_ string getUserName ()

  Returns the username parameter as a string or NOTHING if none is set.

- bool inTransaction ()

  Returns True if a transaction is currently in progress (meaning in this case that a datasource form the pool is dedicated to the calling thread), False if not.

- nothing rollback ()

  Rolls back the current transaction and releases the connection to the pool.

- any select (string sql,...)

  Executes an SQL select statement on the server and returns the result as a hash (column names) of lists (column values per row)

- any selectRow (string sql,...)

  Executes an SQL select statement on the server and returns the first row as a hash (the column values)

- any selectRows (string sql,...)

  Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

- setErrorTimeout (timeout ts)

  Sets the timeout period for waiting for a connection to come free; note that timeout values less than or equal to zero mean that requests for a connection will wait indefinitely.

- nothing setEventQueue (Qore::Thread::Queue queue, any arg)

  Sets a queue object for DBI events on the pool.
- setWarningCallback (timeout ms, code callback, any arg)

  sets a connection delay warning callback to be called any time the delay assigning a connection from the pool exceeds the given timeout in milliseconds
- string toString ()

  Returns a string with technical information about the object.
- any vexec (string sql, __7_ softlist vargs)

  Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes SQL code on the DB connection, taking a list for all bind arguments.
- any vselect (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.
- any vselectRow (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.
- any vselectRows (string sql, __7_ softlist vargs)

  Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

45.14.1 Detailed Description

Provides transparent per-thread, per-transaction datasource connection pooling.

Restrictions:

Qore::PO_NO_DATABASE

In most cases, the DatasourcePool class can be used as a drop-in replacement for the Datasource class with autocommit disabled; when a transaction begins, a datasource will be automatically assigned to the calling thread, and it will only be released when a commit or rollback is called on the object. If no datasource is available, the calling thread will block until a datasource comes available.

Note that the same principles apply to SQL and database driver usage as with the Datasource class, see the Datasource class documentation for more information.

The DatasourcePool class uses Qore's thread resource tracking infrastructure to raise an exception if a thread terminates while a connection is allocated to it. If Qore user code enters a transaction with a DatasourcePool object and the thread terminates without closing the transaction (via DatasourcePool::commit() or Datasource↩ Pool::rollback()), an exception will automatically be raised, the transaction will be rolled back, and the Datasource connection will be freed to the pool.

DatasourcePool Connection Allocations

The following methods allocate a persistent connection to the calling thread:

- DatasourcePool::exec()
- DatasourcePool::vexec()
- DatasourcePool::execRaw()
- DatasourcePool::beginTransaction()

The connection is released to the pool when DatasourcePool::commit() or DatasourcePool::rollback() are called (or in the case the thread terminates, in which case an exception is raised as well).

To begin a transaction with one of the select methods (for example, with `"select for update"`), call DatasourcePool::beginTransaction() first to manually dedicate a Datasource to the thread before calling the select method. Otherwise statements that should be in the same transaction may be executed in different connections.

Executing a DatasourcePool method while not in a transaction is realized by allocating a temporary connection to the calling thread which is re-released when the method returns. No explicit commits are executed by the class, therefore it is an error to execute transaction-relevant commands without first calling DatasourcePool::exec(), DatasourcePool::vexec(), DatasourcePool::execRaw(), or DatasourcePool::beginTransaction().

Note that the SQLStatement class also grabs allocates a persistent connection to the calling thread when executing if it is created using a DatasourcePool object in the constructor; for more information see the SQLStatement class.

Note

This class is not available with the PO_NO_DATABASE parse option

See also

`SqlUtil` for a high level database-independent API

45.14.2 Member Function Documentation

45.14.2.1 nothing Qore::SQL::DatasourcePool::beginTransaction ( ) [virtual]

Manually allocates a persistent connection from the pool to the calling thread.

This method should be called when a transaction will be started with a DatasourcePool::select() method (or DatasourcePool::vselect(), etc).

This method does not make any communication with the server to start a transaction; it only allocates a persistent connection to the current thread in Qore.

To release the connection, call DatasourcePool::commit() or DatasourcePool::rollback()

Example:

```
$db.beginTransaction();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.2 nothing Qore::SQL::DatasourcePool::clearEventQueue ( )

Clears the queue object for DBI events on the pool.

Since

Qore 0.8.9

45.14.2.3 Qore::SQL::DatasourcePool::clearWarningCallback ( )

clears any connection delay warning callback from the object

Example:

```
$ds.clearWarningCallback();
```

Since

Qore 0.8.9

45.14.2.4 nothing Qore::SQL::DatasourcePool::commit ( ) [virtual]

Commits the current transaction and releases the connection to the pool.

Example:

```
$db.commit();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.5 Qore::SQL::DatasourcePool::constructor (string *driver*, __7_ string *user*, __7_ string *pass*, __7_ string *db*, __7_ string *encoding*, __7_ string *host*, softint *min* = 3, softint *max* = 10, softint *port* = 0, __7_ Qore::Thread::Queue *queue*, any *arg*)

Creates the DatasourcePool object; attempts to load a DBI driver if the driver is not already present in Qore.

Parameters

driver	The name of the DBI driver for the Datasource. See SQL Constants for builtin constants for DBI drivers shipped with Qore, or see the DBI driver documentation to use an add-on driver (this string should be the name of the driver to be loaded)
user	The user name for the new connection. Also see Datasource::setUserName() for a method that allows this parameter to be set after the constructor.
pass	The password for the new connection. Also see Datasource::setPassword() for a method that allows this parameter to be set after the constructor.
db	The database name for the new connection. Also see Datasource::setDBName() for a method that allows this parameter to be set after the constructor.
encoding	The database-specific name of the character encoding to use for the new connection. Also see Datasource::setDBCharset() for a method that allows this parameter to be set after the constructor. If no value is passed for this parameter, then the database character encoding corresponding to the default character encoding will be used instead.
host	The host name for the new connection. Also see Datasource::setHostName() for a method that allows this parameter to be set after the constructor.
min	The minimum number of connections in the pool (this number of connections is opened in the constructor)
max	The maximum number of connections in the pool (not more than this number of connections will be opened)
port	The port number for the new connection. Also see Datasource::setPort() for a method that allows this parameter to be set after the constructor.
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the arg key of datasource events

Example:

```
my DatasourcePool $db(DSPGSQL, "user", "pass", "database", "utf8", "localhost", 3, 10, 5432);
```

Exceptions

DATASOURCEPOOL-UNSUPPORTED-DATABASE	DBI driver cannot be found
DATASOURCEPOOL-CONSTRUCTOR-ERROR	invalid min, max, or port argument

DBI-OPTION-ERROR	unknown or unsupported option passed to driver

45.14.2.6 Qore::SQL::DatasourcePool::constructor (**string** *desc*, __7_ **Qore::Thread::Queue** *queue*, **any** *arg*)

Creates a DatasourcePool object from a single string giving all parameters that can be parsed by parse_datasource()

Parameters

desc	a datasource description string in the format that can be parsed by parse_datasource()
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a `QUEUE-ERROR` will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the `arg` key of datasource events

Example:

```
my DatasourcePool ds("pgsql:user/pass@db01(utf8)%localhost:5432");
```

Exceptions

DATASOURCE-UNSUPPORTED-DATABASE	DBI driver cannot be loaded
DATASOURCE-CONSTRUCTOR-ERRO	missing required parameter for connection; port value is <= 0
DBI-OPTION-ERROR	unknown or unsupported option passed to driver

Since

Qore 0.8.6

45.14.2.7 Qore::SQL::DatasourcePool::constructor (**hash** *opts*, __7_ **Qore::Thread::Queue** *queue*, **any** *arg*)

Creates a Datasource object from a hash argument giving parameters for the constructor.

Parameters

opts	a hash giving parameters for the new datasource with the following possible keys (the "type" key is mandatory, also usable with the output of the parse_datasource() function): • type: (*string) The name of the database driver to use; this key is mandatory; if not present, an exception will be raised. See SQL Constants for builtin constants for DBI drivers shipped with Qore, or see the DBI driver documentation to use an add-on driver (this string should be the name of the driver to be loaded) • user: (*string) The user name for the new connection • pass: (*string) The password for the new connection • db: (*string) The database name for the new connection • charset: (*string) The database-specific name of the character encoding to use for the new connection. Also see DatasourcePool::setDBCharset() for a method that allows this parameter to be set after the constructor. If no value is passed for this parameter, then the database character encoding corresponding to the default character encoding for the Qore process will be used instead. • host: (*string) The host name for the new connection • port: (softint) The port number for the new connection • options: (hash) An optional hash having "min" and "max" keys giving the minimum and maximum number of connections in the pool, respectively; all other options will be passed to the database driver
queue	An optional Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown; passing NOTHING will clear any event queue
arg	an optional argument to be included in the arg key of datasource events

Example:

```
my Datasource $db(("type": DSPGSQL, "user": "username", "pass": "password", "db": "database", "
      charset": "utf8", "host": "localhost", "port": 5432, "options": ("min": 3, "max": 10)));
```

Exceptions

DATASOURCEPOOL-UNSUPPORTED-DATABASE	DBI driver cannot be loaded
DATASOURCEPOOL-CONSTRUCTOR-ERROR	missing or invalid "driver" key, other key name not assigned to a string; "port" value is <= 0; invalid "min" or "max" keys
DBI-OPTION-ERROR	unknown or unsupported option passed to driver

45.14.2.8 Qore::SQL::DatasourcePool::copy ( )

Creates a new Datasource object with the same driver as the original and copies of all the connection parameters.

Example:

```
my DatasourcePool $new_dsp = $dsp.copy();
```

45.14.2.9 bool Qore::SQL::DatasourcePool::currentThreadInTransaction ( )

Returns True if the current thread is in a transaction (i.e. has a dedicated datasource allocation), False if not.

Returns

True if the current thread is in a transaction (i.e. has a dedicated datasource allocation), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $db.currentThreadInTransaction();
```

Since

Qore 0.8.7

45.14.2.10 Qore::SQL::DatasourcePool::destructor ( )

Throws an exception if any transactions are in progress and returns immediately; the object is destroyed after any in-progress requests are completed.

Example:

```
delete $db;
```

Exceptions

DATASOURCEPOOL-ERROR	The destructor was called while a transaction was still in progress

45.14.2.11 any Qore::SQL::DatasourcePool::exec (string *sql*, ...) [virtual]

Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes an SQL command on the server and returns either the integer row count (for example, for updates, inserts, and deletes) or the data retrieved (for example, if a stored procedure is executed that returns values)

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.exec("insert into table (varchar_col, timestamp_col, blob_col, numeric_col) values (%v,
       %v, %v, %d)", $string, now(), $binary, 100);
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.12 any Qore::SQL::DatasourcePool::execRaw (string *sql*) [virtual]

Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes an SQL command on the server and returns either the row count (for example, for updates and inserts) or the data retrieved (for example, if a stored procedure is executed that returns values)

This method does not do any variable binding, so it's useful for example for DDL statements etc

Warning:

Using this method to execute pure dynamic SQL many times with different SQL strings (as opposed to using the same string and binding by value instead of dynamic SQL) can affect application performance by prohibiting the efficient usage of the DB server's statement cache. See DB server documentation for variable binding and the SQL statement cache for more information.

Parameters

sql	The SQL command to execute on the server; this string will not be subjected to any transformations for variable binding

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
$db.execRaw("create table my_tab (id number, some_text varchar2(30))");
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.13 any Qore::SQL::DatasourcePool::getClientVersion ( ) [virtual]

Retrieves the driver-specific client library version information; this method may not be implemented for all drivers.

Returns

the driver-specific client library version information; this method may not be implemented for all drivers; see the DBI driver documentation for the return data type and format

Example:

```
my any $ver = $db.getClientVersion();
```

Note

see the documentation for the DBI driver being used for possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.14 hash Qore::SQL::DatasourcePool::getConfigHash ( ) [virtual]

Returns a datasource hash describing the configuration of the current object.

Code Flags:

CONSTANT

Example:

```
my hash $h = $ds.getConfigHash();
```

Returns

a datasource hash describing the configuration of the current object

Since

Qore 0.8.8

Implements Qore::SQL::AbstractDatasource.

45.14.2.15 string Qore::SQL::DatasourcePool::getConfigString ( ) [virtual]

Returns a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Code Flags:

CONSTANT

Example:

```
my string $str = $ds.getConfigString();
```

Returns

a string giving the configuration of the current object in a format that can be parsed by parse_datasource()

Since

Qore 0.8.8

Implements Qore::SQL::AbstractDatasource.

45.14.2.16 __7_ string Qore::SQL::DatasourcePool::getDBCharset ( )

Retrieves the database-specific charset set encoding for the object.

A method synonym for DatasourcePool::getDBEncoding() kept for backwards-compatibility

Returns

the database-specific charset set encoding for the object

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getDBCharset();
```

45.14.2.17 string Qore::SQL::DatasourcePool::getDBEncoding ( ) [virtual]

Retrieves the database-specific charset set encoding for the object.

Returns

the database-specific charset set encoding for the object

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getDBEncoding();
```

See also

DatasourcePool::getOSEncoding()

Implements Qore::SQL::AbstractDatasource.

45.14.2.18 __7_ string Qore::SQL::DatasourcePool::getDBName ( ) [virtual]

Returns the database name parameter as a string or NOTHING if none is set.

Returns

the database name parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $db = $db.getDBName();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.19 string Qore::SQL::DatasourcePool::getDriverName ( ) [virtual]

Returns the name of the driver used for the object.

Returns

the name of the driver used for the object

Code Flags:

CONSTANT

Example:

```
my string $driver = $db.getDriverName();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.20 int Qore::SQL::DatasourcePool::getErrorTimeout ( )

Returns the error timeout period for waiting for a connection to come free as an integer giving milliseconds; note that timeout values less than or equal to zero mean that requests for a connection will wait indefinitely.

Code Flags:

CONSTANT

Example:

```
my int $ms = $ds.getErrorTimeout();
```

Returns

the error timeout period for waiting for a connection to come free as an integer giving milliseconds; note that timeout values less than or equal to zero mean that requests for a connection will wait indefinitely

Since

Qore 0.8.9

45.14.2.21 __7_ string Qore::SQL::DatasourcePool::getHostName ( ) [virtual]

Returns the hostname parameter as a string or NOTHING if none is set.

Returns

the hostname parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $host = $db.getHostName();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.22 int Qore::SQL::DatasourcePool::getMaximum ( )

Returns the maximum number of connections in this object.

Returns

the maximum number of connections in this object

Code Flags:

CONSTANT

Example:

```
my int $max = $db.getMaximum();
```

45.14.2.23 int Qore::SQL::DatasourcePool::getMinimum ( )

Returns the minimum number of connections in this object.

Returns

the minimum number of connections in this object

Code Flags:

CONSTANT

Example:

my int $min = $db.getMinimum();

45.14.2.24 any Qore::SQL::DatasourcePool::getOption (string *opt*)

Returns the current value for the given option.

Code Flags:

RET_VALUE_ONLY

Parameters

opt	the option to get

Exceptions

DBI-OPTION-ERROR	unknown or unsupported option passed to driver

Since

Qore 0.8.6

45.14.2.25 hash Qore::SQL::DatasourcePool::getOptionHash ( )

returns the valid options for the driver associated with the Datasource with descriptions and current values for the current Datasource object

Code Flags:

CONSTANT

Returns

a hash where the keys are valid option names, and the values are hashes with the following keys:

- `"desc"`: a string description of the option
- `"type"`: a string giving the data type restriction for the option
- `"value"`: the current value of the option

Since

Qore 0.8.6

45.14.2.26 string Qore::SQL::DatasourcePool::getOSCharset ( )

Returns the Qore character encoding name for the object as a string or " (unknown) " if none is set.

Returns

the Qore character encoding name for the object as a string or " (unknown) " if none is set

Code Flags:

CONSTANT

Example:

```
my string $enc = $db.getOSCharset();
```

See also

DatasourcePool::getOSEncoding()

45.14.2.27 __7_ string Qore::SQL::DatasourcePool::getOSEncoding ( ) [virtual]

Returns the Qore character encoding name for the object as a string or NOTHING if none is set.

Returns

the Qore character encoding name for the object as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $enc = $db.getOSEncoding();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.28 __7_ string Qore::SQL::DatasourcePool::getPassword ( ) [virtual]

Returns the password parameter as a string or NOTHING if none is set.

Returns

the password parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $pass = $db.getPassword();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.29 __7_ int Qore::SQL::DatasourcePool::getPort ( ) [virtual]

Gets the port number that will be used for the next connection to the server.

Invalid port numbers will cause an exception to be thrown when the connection is opened

Code Flags:

CONSTANT

Example:

```
my *int $port = $db.getPort();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.30 any Qore::SQL::DatasourcePool::getServerVersion ( ) [virtual]

Returns the driver-specific server version data for the current connection.

Returns

the driver-specific server version data for the current connection; see the DBI driver documentation for the return data type and format

Example:

```
my any $ver = $db.getServerVersion();
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.31 __7_ hash Qore::SQL::DatasourcePool::getUsageInfo ( )

Returns a hash with usage information about the DatasourcePool object.

Code Flags:

CONSTANT

Example:

```
my hash $h = $ds.getUsageInfo();
```

Returns

a hash with the following keys (note that the `callback`, `timeout`, and optionally `arg` keys are only set if a warning callback is set):

- `callback`: the closure or call reference set as the warning callback
- `timeout`: an integer giving the timeout threshold in milliseconds before the warning callback is executed
- `arg`: an optional argument passed to the warning callback
- `wait_max`: the maximum number of microseconds that threads have had to wait for a free connection
- `stats_reqs`: the total number of requests for connections / transactions on this DatasourcePool
- `stats_hits`: the total number of requests for connections / transactions on this DatasourcePool that did not have to wait for a connection

Note

`wait_max` is reported in microseconds (1 ms = 1000 us) while the warning timeout has a resolution of milliseconds

Since

Qore 0.8.9

45.14.2.32 __7_ string Qore::SQL::DatasourcePool::getUserName ( ) [virtual]

Returns the username parameter as a string or NOTHING if none is set.

Returns

the username parameter as a string or NOTHING if none is set

Code Flags:

CONSTANT

Example:

```
my *string $user = $db.getUserName();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.33 bool Qore::SQL::DatasourcePool::inTransaction ( ) [virtual]

Returns True if a transaction is currently in progress (meaning in this case that a datasource form the pool is dedicated to the calling thread), False if not.

Returns

True if a transaction is currently in progress (meaning in this case that a datasource form the pool is dedicated to the calling thread), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $db.inTransaction();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.34 nothing Qore::SQL::DatasourcePool::rollback ( ) [virtual]

Rolls back the current transaction and releases the connection to the pool.

Example:

```
$db.rollback();
```

Implements Qore::SQL::AbstractDatasource.

45.14.2.35 any Qore::SQL::DatasourcePool::select (string *sql*, ...) [virtual]

Executes an SQL select statement on the server and returns the result as a hash (column names) of lists (column values per row)

The return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

Additionally, this format is a more efficient format than that returned by the Datasource::selectRows() method, because the column names are not repeated for each row returned. Therefore, for retrieving anything greater than small amounts of data, it is recommended to use this method instead of Datasource::selectRows().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method returns a hash (the keys are the column names) of lists (the column data per row) when executed with an SQL select statement, however some DBI drivers allow any SQL to be executed through this method, in which case other data types can be returned (such as an integer for a row count or a hash for output parameters when executing a stored procedure)

Example:

```
# bind a string and a date/time value by value in a query
$query = $db.select("select * from table where varchar_column = %v and timestamp_column > %v", $string,
       2007-10-11T15:31:26.289);
```

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

- DatasourcePool::vselect()
- Qore::zzz8hashzzz9::contextIterator()
- context Statements

Implements Qore::SQL::AbstractDatasource.

45.14.2.36 any Qore::SQL::DatasourcePool::selectRow (string *sql*, ...) [virtual]

Executes an SQL select statement on the server and returns the first row as a hash (the column values)

If more than one row is returned, then it is treated as an error and a `DBI-SELECT-ROW-ERROR` is returned (however the DBI driver should raise its own exception here to avoid retrieving more than one row from the server). For a similar method taking a list for all bind arguments, see DatasourcePool::vselectRow().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as "`select for update`"), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method normally returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $h = $db.selectRow("select * from example_table where id = 1");
```

Exceptions

DBI-SELECT-ROW-ERR↩ *OR*	more than 1 row retrieved from the server

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.37 any Qore::SQL::DatasourcePool::selectRows (**string** *sql*, *...*) [virtual]

Executes an SQL select statement on the server and returns the result as a list (rows) of hashes (the column values)

The return format of this method is not as memory efficient as that returned by the DatasourcePool::select() method, therefore for larger amounts of data, it is recommended to use DatasourcePool::select().

The usual return value of this method can be iterated with the ListHashIterator class.

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as "`select for update`"), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
...	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.selectRows("select * from example_table");
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

See also

DatasourcePool::select()

Implements Qore::SQL::AbstractDatasource.

45.14.2.38 Qore::SQL::DatasourcePool::setErrorTimeout (timeout *ts*)

Sets the timeout period for waiting for a connection to come free; note that timeout values less than or equal to zero mean that requests for a connection will wait indefinitely.

Example:

```
$ds.setErrorTimeout(30s);
```

Since

Qore 0.8.9

45.14.2.39 nothing Qore::SQL::DatasourcePool::setEventQueue (Qore::Thread::Queue *queue*, any *arg*)

Sets a queue object for DBI events on the pool.

Parameters

queue	the Queue object to receive datasource events; note that the Queue passed cannot have any maximum size set or a `QUEUE-ERROR` will be thrown; passing NOTHING will clear any event queue
arg	an argument to be included in the `arg` key of datasource events

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

Since

Qore 0.8.9

45.14.2.40 Qore::SQL::DatasourcePool::setWarningCallback (timeout *ms*, code *callback*, any *arg*)

sets a connection delay warning callback to be called any time the delay assigning a connection from the pool exceeds the given timeout in milliseconds

Example:

```
my code $cb = sub (string $dsstr, int $us, int $to) {
    printf("WARNING: datasource pool %y took %f ms (threshold %d ms) to assign a new connection\n",
      $dsstr, $us.toNumber() / 1000n, $to);
};
$ds.setWarningCallback(5s, $cb);
```

Parameters

ms	the period of time with a resolution of milliseconds after which the callback will be called if a connection cannot be allocated in the given time period
callback	a closure or call reference taking three or four arguments: (string *desc*, int *time*, int *warning_timeout*[, any *arg*]) which will be passed a datasource description string for the pool, an integer giving the amount of time it took to acquire the connection in microseconds (divide by 1000 to get milliseconds), an integer giving the warning timeout threshold in milliseconds, and optionally the *arg* value passed to this method
arg	an optional argument that will be passed to the warning callback

Note

that the warning timeout has a resolution of milliseconds, but the actual wait time is reported in microseconds (1 ms = 1000 us)

Since

Qore 0.8.9

45.14.2.41 string Qore::SQL::DatasourcePool::toString ( )

Returns a string with technical information about the object.

Returns

a string with technical information about the object

Code Flags:

CONSTANT

Example:

```
my string $str = $db.toString();
```

45.14.2.42 any Qore::SQL::DatasourcePool::vexec (string *sql*, __7_ softlist *vargs*) [virtual]

Allocates a persistent connection to the current thread from the pool (if one has not already been allocated) and executes SQL code on the DB connection, taking a list for all bind arguments.

Same as DatasourcePool::exec() except takes an explicit list for bind arguments

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

The return value depends on the DBI driver; normally, for commands with placeholders, a hash is returned holding the values acquired from executing the SQL statement. For all other commands, normally an integer row count is returned. However, some DBI drivers also allow select statements to be executed through this interface, which would also return a hash (column names) of lists (values for each column).

Example:

```
my int $rows = $db.vexec("insert into example_table value (%v, %v, %v)", $arg_list);
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

Implements Qore::SQL::AbstractDatasource.

45.14.2.43 any Qore::SQL::DatasourcePool::vselect (string *sql*, __7_ softlist *vargs*) [virtual]

Executes a select statement on the server and returns the results in a hash (column names) of lists (column values per row), taking a list for all bind arguments.

The return format of this method is suitable for use with context statements, for easy iteration and processing of query results. Alternatively, the HashListIterator class can be used to iterate the return value of this method.

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as `"select for update"`), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a hash (the keys are the column names) of list (each hash key's value is a list giving the row data), however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $query = $db.vselect("select * from example_table where id = %v and name = %v", $arg_list);
```

Note

- See the documentation for the DBI driver being used for additional possible exceptions
- This method returns all the data available immediately; to process query data piecewise, use the SQLStatement class

See also

- DatasourcePool::select()
- Qore::zzz8hashzzz9::contextIterator()
- context Statements

Implements Qore::SQL::AbstractDatasource.

45.14.2.44 any Qore::SQL::DatasourcePool::vselectRow (string *sql*, __7_ softlist *vargs*) [virtual]

Executes a select statement on the server and returns the first row as a hash (column names and values), taking a list for all bind arguments.

This method is the same as the DatasourcePool::selectRow() method, except this method takes a single argument after the SQL command giving the list of bind value parameters

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as "`select for update`"), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

This method normally returns a hash (the keys are the column names) of row data or NOTHING if no row is found for the query when executed with an SQL select statement, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *hash $h = $db.vselectRow("select * from example_table where id = %v and type = %v", $arg_list);
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

See also

DatasourcePool::selectRow()

Implements Qore::SQL::AbstractDatasource.

45.14.2.45 any Qore::SQL::DatasourcePool::vselectRows (**string** *sql,* __7_ **softlist** *vargs*) [virtual]

Executes a select statement on the server and returns the results in a list (rows) of hashes (column names and values), taking a list for all bind arguments.

Same as the Datasource::selectRows() method, except this method takes a single argument after the SQL command giving the list of bind value parameters.

The usual return value of this method can be iterated with the ListHashIterator class.

The return format of this method is not as memory efficient as that returned by the DatasourcePool::select() method, therefore for larger amounts of data, it is recommended to use DatasourcePool::select().

This method also accepts all bind parameters (`%d`, `%v`, `%s`, etc) as documented in Binding by Value and Placeholder

This method does not retain the datasource connection for the current thread if one was not already allocated before this method is called, so to execute select statements that begin a transaction (such as "`select for update`"), execute DatasourcePool::beginTransaction() first to ensure that the connection is dedicated to the calling thread.

Parameters

sql	The SQL command to execute on the server
vargs	Include any values to be bound (using `%v` in the command string) or placeholder specifications (using **:** *key_name* in the command string) in order after the command string

Returns

Normally returns a list (rows) of hash (where the keys are the column names of each row) or NOTHING if no rows are found for the query, however some DBI drivers allow any SQL statement to be executed through this method (not only select statements), in this case other data types can be returned

Example:

```
my *list $list = $db.vselectRows("select * from example_table where id = %v and type = %v", $arg_list);
```

Note

see the documentation for the DBI driver being used for additional possible exceptions

See also

DatasourcePool::selectRows()

Implements Qore::SQL::AbstractDatasource.

45.15 Qore::Dir Class Reference

This class implements directory handling, file listing, creating/removing subdirectories, etc.

Public Member Functions

- bool chdir (string path)

  Changes the current directory of the Dir object to the path given.
- nothing chgrp (int gid)

  Change the group membership of the directory from the group id.
- nothing chgrp (string groupname)

  Change the group membership of the directory.
- nothing chmod (softint mode)

  Changes the mode of the directory.
- nothing chown (int uid)

  Change the ownership of the directory from the userid.
- nothing chown (string username)

  Change the ownership of the directory from the username.
- constructor (__7_ string encoding)

  Creates the Directory object.
- copy ()

  Creates a new directory object with the same character encoding specification and the same path as the original.
- int create (softint mode=0777)

  Creates the directory tree the Dir object points to, if it does not exist.
- bool exists ()

  Returns True if the path in the Dir object points to a directory that already exists and is openable by the current user, False otherwise.
- hash hstat ()

  Returns a hash of file status information for the current directory.
- list list (bool full=False)

  Get all entries in this directory, except "." and ".." directories; if any errors occur an exception is thrown.
- list list (string regex, softint regex_options=0, softbool full=False)

Gets all entries in the directory that match the given regular expression (except "." and ".." directories); if any errors occur an exception is thrown.

- list listDirs (bool full=False)

  Retrieves all subdirectory entries in this directory, except "." and ".." directories; if any errors occur an exception is thrown.

- list listDirs (string regex, softint regex_options=0, softbool full=False)

  Gets all subdirectory entries in the directory that match the given regular expression (except "." and ".." directories); if any errors occur an exception is thrown.

- list listFiles (bool full=False)

  Retrieves all files in this directory; if any errors occur an exception is thrown.

- list listFiles (string regex, softint regex_options=0, softbool full=False)

  Retrieves all files in the directory that match the given regular expression; if any errors occur an exception is thrown.

- nothing mkdir (string subdir, softint mode=0777)

  Creates a direct subdirectory in the Dir object's current path.

- Dir openDir (string subdir, __7_ string encoding)

  Get a Dir object as an subdir entry of the current directory.

- File openFile (string filename, int flags=O_RDONLY, int mode=0666, __7_ string encoding)

  Create and open a File object in the current directory of the Dir object.

- __7_ string path ()

  Returns the path of the Dir object or NOTHING if no path is set.

- bool removeFile (string file)

  Remove the file with the given name in the Dir object's directory.

- nothing rmdir (string subdir)

  Removes a direct subdirectory from the Dir object's current path.

- list stat ()

  Returns a list of file status information for the current directory.

- hash statvfs ()

  Returns a hash of filesystem status values for the current directory.

45.15.1 Detailed Description

This class implements directory handling, file listing, creating/removing subdirectories, etc.

Restrictions:

Qore::PO_NO_FILESYSTEM

The Dir class allows Qore programs to list and manipulate directories.

Directory objects can be created/opened with a specific character encoding. This means that any entry read from the directory will be tagged with the directory's character encoding. If no character encoding is given the default character encoding is assumed.

Note

This class is not available with the PO_NO_FILESYSTEM parse option

45.15.2 Member Function Documentation

45.15.2.1 bool Qore::Dir::chdir (string *path*)

Changes the current directory of the Dir object to the path given.

If this method returns False then the directory can be created by calling Dir::create()

Parameters

path	The path can be either an absolute path (leading with '/') or a directory relative to the actual path

Returns

True if the new path is openable as directory (see Dir::exists())

Example:

```
if (!$d.chdir("../doc")) {
    $d.create(0755);
}
```

Exceptions

DIR-CHDIR-ERROR	cannot change to relative directory because no current directory is set

45.15.2.2 nothing Qore::Dir::chgrp (int *gid*)

Change the group membership of the directory from the group id.

Parameters

gid	the groupid of the user to change the directory to

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Example:

```
try {
    $dir.chgrp($gid);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CHOWN-ERROR	cannot change directory owner; no directory is set
DIR-CHOWN-FAILURE	error changing directory owner (`arg` will be assigned to the errno value)

45.15.2.3 nothing Qore::Dir::chgrp (string *groupname*)

Change the group membership of the directory.

Parameters

groupname	the group name of the group to change the directory to

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Example:

```
try {
```

```
    $dir.chgrp($groupname);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CHGRP-PARAMETER-ERROR	no userid found for user
DIR-CHOWN-ERROR	cannot change directory owner; no directory is set
DIR-CHOWN-FAILURE	error changing directory owner (`arg` will be assigned to the errno value)

45.15.2.4 nothing Qore::Dir::chmod (softint *mode*)

Changes the mode of the directory.

Parameters

mode	The mode of the directory

Example:

```
try {
    $dir.chmod(0750);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CHMOD-ERROR	cannot change directory's mode; no directory is set
DIR-CHMOD-FAILURE	error changing directory's mode (`arg` will be assigned to the errno value)

45.15.2.5 nothing Qore::Dir::chown (int *uid*)

Change the ownership of the directory from the userid.

Parameters

uid	the userid of the user to change the directory to

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Example:

```
try {
    $dir.chown($uid);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CHOWN-ERROR	cannot change directory owner; no directory is set
DIR-CHOWN-FAILURE	error changing directory owner (`arg` will be assigned to the errno value)

45.15.2.6 nothing Qore::Dir::chown (**string** *username*)

Change the ownership of the directory from the username.

Parameters

username	the username of the user to change the directory to

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Example:

```
try {
    $dir.chown($username);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CHOWN-PARAMETER-ERROR	no userid found for user
DIR-CHOWN-ERROR	cannot change directory owner; no directory is set
DIR-CHOWN-FAILURE	error changing directory owner (`arg` will be assigned to the errno value)

45.15.2.7 Qore::Dir::constructor (__7_ **string** *encoding*)

Creates the Directory object.

The object will point to the currect directory of the process

Parameters

encoding	the name of the default character encoding for filenames retrieved; if this argument is not given, filename strings retrieved will be tagged with the default character encoding of the process

Example:

```
my Dir $dir("utf-8");
```

45.15.2.8 Qore::Dir::copy ( )

Creates a new directory object with the same character encoding specification and the same path as the original.

Example:

```
my Dir $nd = $dir.copy();
```

45.15.2.9 int Qore::Dir::create (softint *mode* = 0777)

Creates the directory tree the Dir object points to, if it does not exist.

Parameters

mode	The mode of the directory

Returns

the number of directories actually created

Example:

```
try {
    if (!$d.exists()) {
        printf("%y: does not exist; creating...\n", $d.path());
        $cnt = $d.create(0755);
    }
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-CREATE-ERROR	cannot create directory; no directory is set
DIR-CREATE-FAILURE	error creating directory (`arg` will be assigned to the errno value)

45.15.2.10 bool Qore::Dir::exists ( )

Returns True if the path in the Dir object points to a directory that already exists and is openable by the current user, False otherwise.

Returns

True if the path in the Dir object points to a directory that already exists and is openable by the current user, False otherwise

Example:

```
if (!$d.exists())
   printf("%y: does not exist or cannot be opened\n", $d.path());
```

45.15.2.11 hash Qore::Dir::hstat ( )

Returns a hash of file status information for the current directory.

If any errors occur an exception will be thrown

Returns

a Stat Hash giving information about the current directory

Example:

```
try {
    my hash $h = $dir.hstat();
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-HSTAT-ERROR	no directory is set
DIR-HSTAT-FAILURE	error stat'ing directory (`arg` will be assigned to the errno value)

45.15.2.12 list Qore::Dir::list (bool *full* = False)

Get all entries in this directory, except `"."` and `".."` directories; if any errors occur an exception is thrown.

Parameters

full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all entries in the directory (except `"."` and `".."` directories); if *full* is True, then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    map printf("entry: %s\n", $1), $d.list();
}
catch (hash $ex) {
    stderr.printf("%s: %s", $ex.err, $ex.desc);
    if ($ex.arg)
        stderr.printf(": %s", strerror($ex.arg));
    stderr.printf("\n");
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (`arg` will be assigned to the errno value)

Since

Qore 0.8.8 added the *full* parameter

45.15.2.13 list Qore::Dir::list (string *regex*, softint *regex_options* = `0`, softbool *full* = False)

Gets all entries in the directory that match the given regular expression (except `"."` and `".."` directories); if any errors occur an exception is thrown.

Parameters

regex	a regular expression string used to filter the arguments (note that this is not a glob, but rather a regular expression string)
regex_options	optional bitwise-or'ed regex option constants
full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all entries in the directory that match the given regular expression (except "." and ".." directories); if *full* is True, then the return value is a list of file status value hashes plus a "name" key with the file or directory name and optionally a "link" key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    foreach my string $e in ($d.list("\\.txt$")) {
        printf("entry: %s\n");
    }
}
catch (hash $ex) {
    stderr.printf("%s: %s", $ex.err, $ex.desc);
    if ($ex.arg)
        stderr.printf(": %s", strerror($ex.arg));
    stderr.printf("\n");
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (arg will be assigned to the errno value)
REGEX-COMPILATION-ERROR	error in regular expression
REGEX-OPTION-ERROR	regex option argument contains invalid option bits

Since

Qore 0.8.8 added the *full* parameter

45.15.2.14 list Qore::Dir::listDirs (bool *full* = False)

Retrieves all subdirectory entries in this directory, except "." and ".." directories; if any errors occur an exception is thrown.

Parameters

full	if True then the return value is a list of file status value hashes plus a "name" key with the file or directory name and optionally a "link" key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all subdirectory entries in this directory, except "." and ".." directories; if *full* is True, then the return value is a list of file status value hashes plus a "name" key with the file or directory name and optionally a "link" key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    foreach my string $e in ($d.listDirs()) {
        printf("entry: %s\n");
    }
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (`arg` will be assigned to the errno value)

Since

Qore 0.8.8 added the *full* parameter

45.15.2.15 list Qore::Dir::listDirs (string *regex*, softint *regex_options* = `0`, softbool *full* = False)

Gets all subdirectory entries in the directory that match the given regular expression (except `"."` and `".."` directories); if any errors occur an exception is thrown.

Parameters

regex	a regular expression string used to filter the arguments (note that this is not a glob, but rather a regular expression string)
regex_options	optional bitwise-or'ed regex option constants
full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all subdirectory entries in the directory that match the given regular expression (except `"."` and `".."` directories); if *full* is True, then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    foreach my string $e in ($d.listDirs("^pgsql-")) {
        printf("entry: %s\n");
    }
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (`arg` will be assigned to the errno value)
REGEX-COMPILATION-ERROR	error in regular expression
REGEX-OPTION-ERROR	regex option argument contains invalid option bits

Since

Qore 0.8.8 added the *full* parameter

45.15.2.16 list Qore::Dir::listFiles (bool *full* = False)

Retrieves all files in this directory; if any errors occur an exception is thrown.

Parameters

full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name, otherwise a simple list of string names is returned
full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all files in the current directory of the object; if *full* is True, then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    foreach my string $e in ($d.listFiles()) {
        printf("entry: %s\n");
    }
}
catch (hash $ex) {
    stderr.printf("%s: %s", $ex.err, $ex.desc);
    if ($ex.arg)
        stderr.printf(": %s", strerror($ex.arg));
    stderr.printf("\n");
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (`arg` will be assigned to the errno value)

Since

Qore 0.8.8 added the *full* parameter

45.15.2.17 list Qore::Dir::listFiles (string *regex*, softint *regex_options* = `0`, softbool *full* = False)

Retrieves all files in the directory that match the given regular expression; if any errors occur an exception is thrown.

Parameters

regex	a regular expression string used to filter the arguments (note that this is not a glob, but rather a regular expression string)
regex_options	optional bitwise-or'ed regex option constants
full	if True then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Returns

a list of all files in the directory that match the given regular expression; if *full* is True, then the return value is a list of file status value hashes plus a `"name"` key with the file or directory name and optionally a `"link"` key for symbolic link targets, otherwise a simple list of string names is returned

Example:

```
try {
    foreach my string $e in ($d.listFiles("\\.txt$")) {
        printf("entry: %s\n");
    }
}
catch (hash $ex) {
```

```
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-READ-ERROR	no directory is set
DIR-READ-FAILURE	error reading directory (`arg` will be assigned to the errno value)
REGEX-COMPILATION-ERROR	error in regular expression
REGEX-OPTION-ERROR	regex option argument contains invalid option bits

Since

Qore 0.8.8 added the *full* parameter

45.15.2.18 nothing Qore::Dir::mkdir (string *subdir*, softint *mode* = `0777`)

Creates a direct subdirectory in the Dir object's current path.

Parameters

subdir	The subdirectory name to create; only direct subdirectories are allowed; directory separator characters are not allowed
mode	The mode of the directory

Example:

```
try {
    $dir.mkdir("subdir", 0750);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-MKDIR-PARAMETER-ERROR	only direct subdirectories are allowed
DIR-MKDIR-FAILURE	error creating directory (`arg` will be assigned to the errno value)

45.15.2.19 Dir Qore::Dir::openDir (string *subdir*, *__7_* string *encoding*)

Get a Dir object as an subdir entry of the current directory.

Parameters

subdir	The name of the subdirectory for the new Dir object (which must be in the current directory of the current Dir object; no path separator characters are allowed)
encoding	The name of the default character encoding for the new Dir object; if this argument is not given, the new Dir object will be tagged with the character encoding of the current Dir object

Returns

The Dir object created for the directory

Example:

```
my Dir $sd = $d.openDir("mysubdir");
```

Exceptions

DIR-OPENDIR-PARAMETER-ERROR	only direct subdirectory names without path separators are allowed

45.15.2.20 File Qore::Dir::openFile (string *filename*, int *flags* = O_RDONLY, int *mode* = 0666, __7_ string *encoding*)

Create and open a File object in the current directory of the Dir object.

This method uses the File::open2() method to open the file.

Parameters

filename	The filename for the file which must be in the current directory (no path separator characters are allowed)
flags	Flags that determine the way the file is accessed, see File Open Constants for more information
mode	Permission bits for when the file is to be created
encoding	The name of the default character encoding for this file; if this argument is not given, the file will be tagged with the default character encoding for the process

Example:

```
# open a file for writing in the directory and set the mode to 0644 and the encoding to UTF-8
my File $f = $d.openFile("myfile.txt", O_CREAT|O_WRONLY, 0644, "utf-8");
```

Exceptions

DIR-OPENFILE-PARAMETER-ERROR	only direct subdirectory names without path separators are allowed

Note

see File::open2() for additional exceptions that can be thrown opening the file

45.15.2.21 __7_ string Qore::Dir::path ( )

Returns the path of the Dir object or NOTHING if no path is set.

This path does not necessarily need to exist; the path is adjusted to remove "." and ".." from the path if present

Returns

the path of the Dir object or NOTHING if no path is set

Example:

```
my *string $mypath = $d.path();
```

45.15.2.22 bool Qore::Dir::removeFile (string *file*)

Remove the file with the given name in the Dir object's directory.

If any errors occur unlinking the file, then an exception occurs

Parameters

file	Remove the file with the given name in the Dir object's directory

Returns

True if the file was present and could be removed, False if the file did not exist

Example:

```
try {
    my bool $b = $dir.removeFile($filename);
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-REMOVEFILE-PARAMETER-ERROR	only filenames without path (i.e. without 'c' characters) are allowed
DIR-REMOVEFILE-FAILURE	the unlink() function returned an error (`arg` will be assigned to the errno value)

45.15.2.23 nothing Qore::Dir::rmdir (string *subdir*)

Removes a direct subdirectory from the Dir object's current path.

Parameters

subdir	The subdirectory name to remove; only direct subdirectories are allowed; directory separator characters are not allowed

Example:

```
try {
    $dir.rmdir("subdir");
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-RMDIR-PARAMETER-ERROR	only direct subdirectories are allowed
DIR-RMDIR-FAILURE	error removing directory (`arg` will be assigned to the errno value)

45.15.2.24 list Qore::Dir::stat ( )

Returns a list of file status information for the current directory.

If any errors occur an exception will be thrown

Returns

a Stat List giving information about the current directory

Example:

```
try {
    my list $l = $dir.stat();
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

Exceptions

DIR-STAT-ERROR	no directory is set
DIR-STAT-FAILURE	error stat'ing directory (`arg` will be assigned to the errno value)

45.15.2.25 hash Qore::Dir::statvfs ( )

Returns a hash of filesystem status values for the current directory.

Returns

a hash of filesystem status values for the current directory

Platform Availability:

Qore::Option::HAVE_STATVFS

Example:

```
try {
    my hash $h = $dir.statvfs();
}
catch (hash $ex) {
    if ($ex.arg)
        printf("%s: %s: %s", $ex.err, $ex.desc, strerror($ex.arg));
    else
        printf("%s: %s", $ex.err, $ex.desc);
}
```

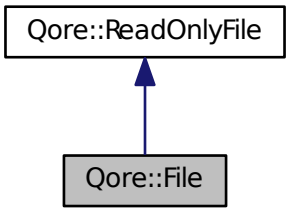
Exceptions

DIR-STATVFS-ERROR	no directory is set
DIR-STATVFS-FAILURE	error in statvfs (`arg` will be assigned to the errno value)

45.16 Qore::File Class Reference

The File class allows Qore programs to read, write, and create files.

Inheritance diagram for Qore::File:



Public Member Functions

- nothing chown (softint uid, softint gid=-1)

  Changes the user and group owners of the file on the filesystem (if the current user has sufficient permission to do so)
- constructor (__7_ string encoding)

  Creates the File object.
- copy ()

  Creates a new File object with the same character encoding specification as the original, otherwise no other information is copied.
- destructor ()

  Closes the File if it is open and destroys the File object.
- int f_printf (string fmt,...)

  Writes a formatted string with hard field widths to the file.
- int f_printf ()

  This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int f_vprintf (string fmt, any fmt_args)

  Writes a formatted string with hard field widths to a file, where the second argument is the formatting argument list.
- int f_vprintf ()

  This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- string getCharset ()

  Returns the character encoding for the File.
- hash getLockInfo ()

  Returns a hash of lock information.
- nothing getTerminalAttributes (TermIOS termios)

  Saves the current terminal attributes for the file in the TermIOS object passed; changes the object passed as an argument to reflect the terminal attributes as set for the File.
- TermIOS getTerminalAttributes ()

  Returns the current terminal attributes for the file as a TermIOS object returned as the return value.
- int lock (softint type=F_RDLCK, softint start=0, softint len=0, softint whence=SEEK_SET)

  Attempts to lock the file according to the arguments passed, does not block.
- nothing lockBlocking (softint type=F_RDLCK, softint start=0, softint len=0, softint whence=SEEK_SET)

  Attempts to lock the file according to the arguments passed, blocking.
- int open (string path, softint flags=O_RDONLY, softint mode=0666, __7_ string encoding)

Opens a File in a particular mode, returns an error code on failure.

- nothing open2 (string path, softint flags=O_RDONLY, softint mode=0666, __7_ string encoding)

  Opens a file in a particular mode; throws an exception on failure.
- int print (string data)

  Writes string data to a file; string data is converted to the File's character encoding if necessary before writing.
- int printf (string fmt,...)

  Writes a formatted string with soft field widths to the file.
- int printf ()

  This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- nothing setCharset (__7_ string encoding)

  Sets the character encoding for the file; if called with no argument, the default encoding is set.
- nothing setTerminalAttributes (softint action=TCSANOW, TermIOS termios)

  Sets the current terminal attributes for the File from the TermIOS object passed; does not change the object passed.
- int sync ()

  Flushes the file's buffer to disk.
- int vprintf (string fmt, any fmt_args)

  Writes a formatted string with soft field widths to a file, where the second argument is the formatting argument list.
- int vprintf ()

  This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.
- int write (binary data)

  Writes binary data to a file.
- int write (string data)

  Writes string data to a file; string data is converted to the File's character encoding if necessary before writing.
- int writei1 (int c)

  Writes a 1-byte integer to the file.
- int writei2 (int s)

  Writes a 2-byte (16 bit) integer to the file in binary big-endian format.
- int writei2LSB (int s)

  Writes a 2-byte (16 bit) integer to the file in binary little-endian format.
- int writei4 (int i)

  Writes a 4-byte (32 bit) integer to the file in binary big-endian format.
- int writei4LSB (int i)

  Writes a 4-byte (32 bit) integer to the file in binary little-endian format.
- int writei8 (int i)

  Writes an 8-byte (64 bit) integer to the file in binary big-endian format.
- int writei8LSB (int i)

  Writes an 8-byte (64 bit) integer to the file in binary little-endian format.

Static Public Member Functions

- static hash hlstat (string path)

  Returns a Stat Hash about the file's status (does not follow symbolic links) or throws an exception if any errors occur.
- static hash hstat (string path)

  Returns a Stat Hash about the file's status (follows symbolic links) or throws an exception if any errors occur.
- static list lstat (string path)

  Returns a Stat List about the given path's status (does not follow symbolic links) or throws an exception if any errors occur.
- static list stat (string path)

  Returns a Stat List about the file's status (follows symbolic links) or throws an exception if any errors occur.
- static hash statvfs (string path)

  Returns a Filesystem Status Hash about filesystem status of the given path; throws an exception if any errors occur.

45.16.1 Detailed Description

The File class allows Qore programs to read, write, and create files.

Restrictions:

Qore::PO_NO_FILESYSTEM

Note

This class is not available with the PO_NO_FILESYSTEM parse option

File objects are created and/or opened with a specific character encoding, meaning that any string read from the file will be tagged with the file's character encoding, and any string data written to the file will be transparently converted to that character encoding before being written (if necessary). If no character encoding is specified, then the default character encoding is assumed for the file.

This class supports posting events to a Queue. See I/O Event Handling for more information.

The events raised by this object are listed in the following table:

File Events

Name	Description
EVENT_DATA_READ	Raised when data is read from the file
EVENT_DATA_WRITTEN	Raised when data is written to the file
EVENT_CHANNEL_CLOSED	Raised when the file is closed
EVENT_DELETED	Raised when the object being monitored is deleted
EVENT_OPEN_FILE	Raised right before an attempt to open a file is made
EVENT_FILE_OPENED	Raised when the file has been successfully opened

45.16.2 Member Function Documentation

45.16.2.1 nothing Qore::File::chown (softint *uid*, softint *gid* = -1)

Changes the user and group owners of the file on the filesystem (if the current user has sufficient permission to do so)

Platform Availability:

Qore::Option::HAVE_UNIX_FILEMGT

Example:

```
$f.chown(0, 0);
```

Parameters

uid	The user id of the user to change to; -1 means do not change uid
gid	The group id of the user to change to; -1 means do not change gid

Exceptions

FILE-CHOWN-ERROR	File is not open or the chown operation failed
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_UNIX_FILEMGT before calling this method to avoid this exception

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.2 Qore::File::constructor (__7_ string *encoding*)

Creates the File object.

It accepts one optional argument that will set the default character encoding for the file (only affects reading and writing string data) To open the file, call File::open() or File::open2(); the character encoding can also be set or changed by the File::open(), File::open2(), or File::setEncoding() methods.

Example:

```
my File $f();
$f.open2("/tmp/my-file.txt");
```

Parameters

encoding	The character encoding for the File. Any strings written to the File will be converted to this character encoding if necessary; if this argument is not given then the File will receive the default encoding

Exceptions

ILLEGAL-EXPRESSION	File::constructor() cannot be called with a TTY target when %no-terminal-io is set

See also

File::open(), File::open2()

45.16.2.3 Qore::File::copy ( )

Creates a new File object with the same character encoding specification as the original, otherwise no other information is copied.

Example:

```
my File $f1 = $f.copy();
```

45.16.2.4 Qore::File::destructor ( )

Closes the File if it is open and destroys the File object.

Closes the File if it is open and destroys the File object

45.16.2.5 int Qore::File::f_printf (string *fmt*, ...)

Writes a formatted string with hard field widths to the file.

This method does not allow arguments to overrun field width specifiers in the format string.

Example:

```
$f.f_printf("%5s\n", "hello there"); # outputs "hello\n", enforcing the field width
```

Events:

EVENT_DATA_WRITTEN

Parameters

fmt	the format string; see String Formatting for more information about the format string

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::printf() for a similar method that does not enforce field widths

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.6 int Qore::File::f_printf ( )

This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

Returns

a constant 0

45.16.2.7 int Qore::File::f_vprintf (string *fmt*, any *fmt_args*)

Writes a formatted string with hard field widths to a file, where the second argument is the formatting argument list.

This method will not allow arguments to overrun field width specifiers in the format string.

Example:

```
$f.f_vprintf("%5s: %d\n", ("hello there", 2)); # outputs "hello: 2\n", enforcing the field width
```

Events:

EVENT_DATA_WRITTEN

Parameters

fmt	the format string; see String Formatting for more information about the format string

fmt_args	the single argument or list of arguments that will be used as the argument list or the format string. If a single argument is passed instead of a list, it will be used as the first argument as if a list were passed

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::vprintf() for a similar method that does not enforce field widths

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.8 int Qore::File::f_vprintf ( )

This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

Returns

a constant 0

45.16.2.9 string Qore::File::getCharset ( )

Returns the character encoding for the File.

Code Flags:

CONSTANT

A method synonym for Qore::ReadOnlyFile::getEncoding(), kept for backwards-compatibility

Returns

the character encoding for the File

45.16.2.10 hash Qore::File::getLockInfo ( )

Returns a hash of lock information.

Platform Availability:

Qore::Option::HAVE_FILE_LOCKING

Code Flags:

RET_VALUE_ONLY

Example:

```
my hash $hash = $f.getLockInfo();
```

Returns

a hash with the following keys:

- `start`: starting byte of the lock
- `len`: the length in bytes of the locked region
- `pid`: the PID of the process holding the lock
- `type`: see File Locking Constants; if no lock is set on the file, the key type has the value `F_UNLCK`.
- `whence`: always returned as `SEEK_SET` when the call is successful

Exceptions

FILE-LOCK-ERROR	File is not open or the internal fcntl operation failed
MISSING-FEATURE-ERROR	this exception is thrown when the method is not available on the runtime platform; for maximum portability, check the constant Qore::Option::HAVE_FILE_LOCKING before calling this function.
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.11 nothing Qore::File::getTerminalAttributes (TermIOS *termios*)

Saves the current terminal attributes for the file in the TermIOS object passed; changes the object passed as an argument to reflect the terminal attributes as set for the File.

Do not pass a reference to the TermIOS object; pass the object itself as an argument.

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my TermIOS $termios();
stdin.getTerminalAttributes($termios);
```

Parameters

termios	The method writes the current terminal attributes for the file to the object passed

Exceptions

FILE-OPERATION-ERROR	the File is not open
TERMIOS-GET-ERROR	error reading terminal attributes from the file descriptor (ex: not a TTY)
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_TERMIOS before calling this method to avoid this exception
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.12 TermIOS Qore::File::getTerminalAttributes ( )

Returns the current terminal attributes for the file as a TermIOS object returned as the return value.

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my TermIOS $termios = stdin.getTerminalAttributes();
```

Returns

the current terminal attributes for the File

Exceptions

FILE-OPERATION-ERROR	the File is not open
TERMIOS-GET-ERROR	error reading terminal attributes from the file descriptor (ex: not a TTY)
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_TERMIOS before calling this method to avoid this exception
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7

45.16.2.13 static hash Qore::File::hlstat (string *path*) [static]

Returns a Stat Hash about the file's status (does not follow symbolic links) or throws an exception if any errors occur.

Does not follow symbolic links, but rather returns filesystem information for symbolic links. If any errors occur, a `FILE-HSTAT-ERROR` exception is thrown

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my hash $h = File::hstat($path);
```

Parameters

path	the file to stat()

Returns

a Stat Hash about the file's status

Exceptions

FILE-HSTAT-ERROR	stat() call failed

See also

hstat() for a normal function returns NOTHING instead of throwing an exception when errors occur
File Stat Constants

45.16.2.14 static hash Qore::File::hstat (string *path*) [static]

Returns a Stat Hash about the file's status (follows symbolic links) or throws an exception if any errors occur.

This method will follow symbolic links and return information about the target. If any errors occur, a `FILE-HSTAT-ERROR` exception is thrown

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my hash $h = File::hstat($path);
```

Parameters

path	the file to stat()

Returns

a Stat Hash about the file's status

Exceptions

FILE-HSTAT-ERROR	stat() call failed

See also

hstat() for a normal function returns NOTHING instead of throwing an exception when errors occur
File Stat Constants

45.16.2.15 int Qore::File::lock (softint *type* = F_RDLCK, softint *start* = 0, softint *len* = 0, softint *whence* = SEEK_SET)

Attempts to lock the file according to the arguments passed, does not block.

Locks or unlocks a portion of the file or the entire file, for reading or writing, non-blocking. The file must be opened in the appropriate mode before this call or the call will fail with an exception.

Platform Availability:

Qore::Option::HAVE_FILE_LOCKING

Example:

```
# lock the entire file exclusively
$f.lock(F_WRLCK);

# lock a section of the file for reading, start byte 512, 2K range
$f.lock(F_RDLCK, 512, 2048);

# release all locks
$f.lock(F_UNLCK);
```

Parameters

type	Type of lock (or unlock); see File Locking Constants
start	Start byte for lock, 0 is the default (start of file)
len	Length in bytes for range to lock, 0 is the default (rest of file)
whence	Indicates how the relative offset of the file should be calculated for the lock; see File Seek Constants

Returns

0 for success, EACCES if the lock would block (only in the case that the lock would block is no exception thrown and EACCES returned)

Exceptions

FILE-LOCK-ERROR	File is not open, lock length is negative, or the fcntl operation failed
MISSING-FEATURE-ERROR	this exception is thrown when the method is not available on the runtime platform; for maximum portability, check the constant Qore::Option::HAVE_FILE_LOCKING before calling this function.
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::lockBlocking() for a blocking version of this method

45.16.2.16 nothing Qore::File::lockBlocking (softint *type* = F_RDLCK, softint *start* = 0, softint *len* = 0, softint *whence* = SEEK_SET)

Attempts to lock the file according to the arguments passed, blocking.

Locks or unlocks a portion of the file or the entire file, for reading or writing, blocking. The file must be opened in the appropriate mode before this call or the call will fail with an exception.

Platform Availability:

Qore::Option::HAVE_FILE_LOCKING

Example:

```
# lock the entire file exclusively
$f.lockBlocking(F_WRLCK);

# lock a section of the file for reading, start byte 512, 2K range
$f.lockBlocking(F_RDLCK, 512, 2048);

# release all locks
$f.lockBlocking(F_UNLCK);
```

Parameters

type	Type of lock (or unlock); see File Locking Constants
start	Start byte for lock, 0 is the default (start of file)
len	Length in bytes for range to lock, 0 is the default (rest of file)
whence	Indicates how the relative offset of the file should be calculated for the lock; see File Seek Constants

Exceptions

FILE-LOCK-ERROR	File is not open, lock length is negative, or the fcntl operation failed
MISSING-FEATURE-ERROR	this exception is thrown when the method is not available on the runtime platform; for maximum portability, check the constant Qore::Option::HAVE_FILE_LOCKING before calling this function.
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::lockBlocking() for a blocking version of this method

45.16.2.17 static list Qore::File::lstat (string *path*) [static]

Returns a Stat List about the given path's status (does not follow symbolic links) or throws an exception if any errors occur.

Does not follow symbolic links, but rather returns filesystem information for symbolic links. If any errors occur, a `FILE-LSTAT-ERROR` exception is thrown

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my int $mode = File::lstat($path)[2];
```

Parameters

path	the file to stat()

Returns

a list of file status values for the given file

Exceptions

FILE-LSTAT-ERROR	stat() call failed

See also

lstat() for a similar function that does not throw exceptions when errors occur, but rather returns NOTHING
File Stat Constants

45.16.2.18 int Qore::File::open (string *path*, softint *flags* = O_RDONLY, softint *mode* = 0666, *__7_* string *encoding*)

Opens a File in a particular mode, returns an error code on failure.

Opens the File in the mode given; if the File was previously open, it is closed first. Additionally, the file permissions can be given if the file is to be created, and optionally the File's character encoding can be specified.

Note that if no encoding is specified, the File will be tagged with the character encoding set in the File's constructor. Any string data written to the File will be converted to the File's encoding, and any string data read from the File will be automatically tagged with the File's encoding.

Example:

```
# open a file for writing, truncate data if already exists, create the file if doesn't exist
# set 0644 permissions, and convert all string data to ISO-8859-1 encoding
if ($f.open($fn, O_CREAT | O_TRUNC | O_WRONLY, 0644, "ISO-8859-1"))
    printf("%s: %s\n", $fn, strerror(errno()));
```

Events:

EVENT_OPEN_FILE, EVENT_FILE_OPENED

Parameters

path	the path to the file
flags	flags that determine the way the file is accessed, see File Open Constants for more information; if this argument is not given, O_RDONLY will be used as the default value.
mode	permission bits for when the file is to be created (default: 0666)
encoding	the name of the character encoding for the File; if this argument is not given, the file will be tagged with the character encoding given in the constructor

Returns

0 = no error, -1 = see errno() and strerror() for the error message

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::open2() for a version of this method that throws an exception when errors occur opening the file

45.16.2.19 nothing Qore::File::open2 (string *path*, softint *flags* = O_RDONLY, softint *mode* = 0666, *__7_* string *encoding*)

Opens a file in a particular mode; throws an exception on failure.

Opens the file in the mode given; if the File was previously open, it is closed first. Additionally, the file permissions can be given if the file is to be created, and optionally the File's default character encoding can be specified.

Note that if no encoding is specified, the File will be tagged with the character encoding set in the File's constructor. Any string data written to the File will be converted to the File's encoding, and any string data read from the File will be automatically tagged with the File's encoding.

If an error occurs, a FILE-OPEN2-ERROR exception is thrown. For a version of this method that returns an error code, see File::open().

Example:

```
# open a file for writing, truncate data if already exists, create the file if doesn't exist
# set 0644 permissions, and convert all string data to ISO-8859-1 encoding
try {
    $f.open2($fn, O_CREAT | O_TRUNC | O_WRONLY, 0644, "ISO-8859-1");
}
catch (hash $ex) {
    printf("%s: %s: %s\n", $fn, $ex.err, $ex.desc);
}
```

Events:

EVENT_OPEN_FILE, EVENT_FILE_OPENED

Parameters

path	the path to the file
flags	flags that determine the way the file is accessed, see File Open Constants for more information; if this argument is not given, `O_RDONLY` will be used as the default value.
mode	permission bits for when the file is to be created (default: 0666)
encoding	the name of the character encoding for the File; if this argument is not given, the File will be tagged with the character encoding given in the constructor

Exceptions

FILE-OPEN2-ERROR	an error occurred opening the file
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::open() for a version of this method that returns an error code instead of throwing an exception when errors occur opening the File

45.16.2.20 int Qore::File::print (string *data*)

Writes string data to a file; string data is converted to the File's character encoding if necessary before writing.

Example:

```
$f.print($data);
```

Events:

EVENT_DATA_WRITTEN

Parameters

data	the data to be written to the file; string data is converted to the File's character encoding if necessary before writing

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ENCODING-CONVERSION-ERROR	error converting from the string's character encoding to the File's character encoding
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Note

this method is equivalent to File::write(string)

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.21 int Qore::File::printf (string *fmt*, ...)

Writes a formatted string with soft field widths to the file.

This method will allow arguments to overrun field width specifiers in the format string.

Example:

```
$f.printf("%5s\n", "hello there"); # outputs "hello there\n", exceeding field width
```

Events:

EVENT_DATA_WRITTEN

Parameters

fmt	the format string; see String Formatting for more information about the format string

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::f_printf() for a similar method that enforces field widths

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.22 int Qore::File::printf ( )

This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

Returns

a constant 0

45.16.2.23 nothing Qore::File::setCharset (__7_ string *encoding*)

Sets the character encoding for the file; if called with no argument, the default encoding is set.

A method synonym for Qore::ReadOnlyFile::setEncoding(), kept for backwards-compatibility

Parameters

encoding	the character encoding for the file; if called with no argument, the default encoding is set

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.24 nothing Qore::File::setTerminalAttributes (softint *action* = TCSANOW, TermIOS *termios*)

Sets the current terminal attributes for the File from the TermIOS object passed; does not change the object passed.

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my TermIOS $termios();
stdin.getTerminalAttributes($termios);
my TermIOS $orig = $termios.copy();
on_exit
    stdin.setTerminalAttributes(TCSADRAIN, $orig);

my int $lflag = $termios.getLFlag();
$lflag &= ~ICANON;
$lflag &= ~ECHO;
$lflag &= ~ISIG;
$termios.setLFlag($lflag);
$termios.setCC(VMIN, 1);
$termios.setCC(VTIME, 0);
stdin.setTerminalAttributes(TCSADRAIN, $termios);
```

Parameters

action	a binary or'ed value of the following actions: • TCSANOW: the change occurs immediately • TCSADRAIN: the change occurs after all output written to the File has been transmitted to the terminal • TCSAFLUSH: the change occurs after all output written to the File has been transmitted to the terminal • TCSASOFT: the values of the `c_cflag`, `c_ispeed`, and `c_ospeed` fields are ignored

termios	the TermIOS to use

Exceptions

FILE-OPERATION-ERROR	the File is not open
TERMIOS-SET-ERROR	error setting terminal attributes on the file descriptor
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_TERMIOS before calling this method to avoid this exception
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.25 static list Qore::File::stat (string *path*) [static]

Returns a Stat List about the file's status (follows symbolic links) or throws an exception if any errors occur.

This method will follow symbolic links and return information about the target. If any errors occur, a `FILE-STAT-ERROR` exception is thrown

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my int $mode = File::stat($path)[2];
```

Parameters

path	the file to stat()

Returns

a list of file status values for the given file

Exceptions

FILE-STAT-ERROR	stat() call failed

See also

File Stat Constants

45.16.2.26 static hash Qore::File::statvfs (string *path*) [static]

Returns a Filesystem Status Hash about filesystem status of the given path; throws an exception if any errors occur.

If any errors occur, a `FILE-STATVFS-ERROR` exception is thrown

Platform Availability:

Qore::Option::HAVE_STATVFS

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my hash $h = File::statvfs($path);
```

Parameters

path	the path to the filesystem to check

Returns

a Filesystem Status Hash about the filesystem where the file resides

Exceptions

FILE-STATVFS-ERROR	statvfs() call failed
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_STATVFS before calling this method to avoid this exception

45.16.2.27 int Qore::File::sync ( )

Flushes the file's buffer to disk.

Example:

```
if ($f.sync())
    printf("error in File::sync(): %s\n", strerror(errno()));
```

Returns

0 for success, -1 for error (see errno() and strerror() for the error information)

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.16.2.28 int Qore::File::vprintf (string *fmt*, any *fmt_args*)

Writes a formatted string with soft field widths to a file, where the second argument is the formatting argument list.

This method will allow arguments to overrun field width specifiers in the format string.

Example:

```
$f.vprintf("%5s: %d\n", ("hello there", 2)); # outputs "hello there: 2\n", exceeding field width
```

Events:

EVENT_DATA_WRITTEN

Parameters

fmt	the format string; see String Formatting for more information about the format string
fmt_args	the single argument or list of arguments that will be used as the argument list or the format string. If a single argument is passed instead of a list, it will be used as the first argument as if a list were passed

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::f_vprintf() for a similar method that enforces field widths

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.29 int Qore::File::vprintf ( )

This method variant does nothing except return a constant 0; it is only included for backwards-compatibility with qore prior to version 0.8.0 for functions that would ignore type errors in arguments.

Code Flags:

RUNTIME_NOOP

Returns

a constant 0

45.16.2.30 int Qore::File::write (binary *data*)

Writes binary data to a file.

Example:

```
$f.write($data);
```

Events:

EVENT_DATA_WRITTEN

Parameters

data	the data to be written to the file

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.31 int Qore::File::write (string *data*)

Writes string data to a file; string data is converted to the File's character encoding if necessary before writing.

Example:

```
$f.write($data);
```

Events:

EVENT_DATA_WRITTEN

Parameters

data	the data to be written to the file; string data is converted to the File's character encoding if necessary before writing

Returns

the number of bytes written

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ENCODING-CONVERSI↩ ON-ERROR	error converting from the string's character encoding to the File's character encoding
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Note

this variant of this method is equivalent to File::print()

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.32 int Qore::File::writei1 (int *c*)

Writes a 1-byte integer to the file.

Example:

```
$f.writei1($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

c	the integer to write; only the least-significant 8 bits will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.33 int Qore::File::writei2 (int *s*)

Writes a 2-byte (16 bit) integer to the file in binary big-endian format.

Example:

```
$f.writei2($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

s	the integer to write in binary big-endian format; only the least-significant 16 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.34 int Qore::File::writei2LSB (int *s*)

Writes a 2-byte (16 bit) integer to the file in binary little-endian format.

Example:

```
$f.writei2LSB($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

s	the integer to write in binary little-endian format; only the least-significant 16 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.35 int Qore::File::writei4 (int *i*)

Writes a 4-byte (32 bit) integer to the file in binary big-endian format.

Example:

```
$f.writei4($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

i	the integer to write in binary big-endian format; only the least-significant 32 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.36 int Qore::File::writei4LSB (int *i*)

Writes a 4-byte (32 bit) integer to the file in binary little-endian format.

Example:

```
$f.writei4LSB($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

i	the integer to write in binary little-endian format; only the least-significant 32 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.37 int Qore::File::writei8 (int *i*)

Writes an 8-byte (64 bit) integer to the file in binary big-endian format.

Example:

```
$f.writei8($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

i	the integer to write in binary big-endian format; only the least-significant 64 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.16.2.38 int Qore::File::writei8LSB (int *i*)

Writes an 8-byte (64 bit) integer to the file in binary little-endian format.

Example:

```
$f.writei8LSB($val);
```

Events:

EVENT_DATA_WRITTEN

Parameters

i	the integer to write in binary little-endian format; only the least-significant 64 bits of the integer will be written to the file

Returns

0 for success

Exceptions

FILE-WRITE-ERROR	File is not open or an I/O error occurred writing data to the File
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

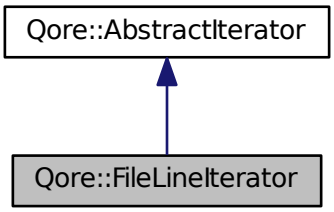
Since

Qore 0.8.7 this method throws exceptions on I/O errors to avoid silent write errors in Qore code

45.17 Qore::FileLineIterator Class Reference

This class defines a line iterator for text files.

Inheritance diagram for Qore::FileLineIterator:



Public Member Functions

- constructor (string path, __7_ string encoding, __7_ string eol, bool trim=True)

  opens the given file for reading with the given options and creates the FileLineIterator object
- copy ()

  Creates a new FileLineIterator object, based on the same object being iterated in the original object (the original file is reopened)
- string getEncoding ()

  Returns the character encoding for the FileLineIterator.
- string getFileName ()

  returns the file path/name used to open the file
- string getLine ()

  returns the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- int getPos ()

  Returns the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)

- string getValue ()

  returns the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- int index ()

  returns the current iterator line number in the file (the first line is line 1) or 0 if not pointing at a valid element
- bool isTty ()

  returns True if the FileLineIterator is connected to a terminal device, False if not
- bool next ()

  Moves the current position to the next line in the file; returns False if there are no more lines to read; if the iterator is not pointing at a valid element before this call, the iterator will be positioned to the beginning of the file.
- reset ()

  Reset the iterator instance to its initial state.
- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.17.1 Detailed Description

This class defines a line iterator for text files.

Since

Qore 0.8.6

Restrictions:

Qore::PO_NO_FILESYSTEM

Example: FileLineIterator basic usage

```
file content:
a2ps-4.13-1332.1.x86_64
a2ps-devel-4.13-1332.1.x86_64
aaa_base-11.3-7.2.x86_64
...

my FileLineIterator $it("/export/home/pvanek/ren.list");
while ($it.next()) {
    printf("%s:%d = %n\n", $it.getFileName(), $it.index(), $it.getValue());
}

/export/home/pvanek/ren.list:1 = "a2ps-4.13-1332.1.x86_64"
/export/home/pvanek/ren.list:2 = "a2ps-devel-4.13-1332.1.x86_64"
/export/home/pvanek/ren.list:3 = "aaa_base-11.3-7.2.x86_64"
...
/export/home/pvanek/ren.list:2155 = "zypper-1.4.5-1.10.x86_64"
```

45.17.2 Member Function Documentation

45.17.2.1 Qore::FileLineIterator::constructor (string *path*, __7_ string *encoding*, __7_ string *eol*, bool *trim* = True)

opens the given file for reading with the given options and creates the FileLineIterator object

Parameters

path	the path to open for reading
encoding	the character encoding tag for underlying file and therefore the string return values for the lines iterated with this class; if not present, the default character encoding is assumed

eol	the optional end of line character(s) to use to detect lines in the file; if this string is not passed, then the end of line character(s) are detected automatically, and can be either "\n", "\r", or "\r\n" (the last one is only automatically detected when not connected to a terminal device in order to keep the I/O from stalling); if this string is passed and has a different character encoding from this object's (as determined by the encoding parameter), then it will be converted to the FileLineIterator's character encoding
trim	if True the string return values for the lines iterated will be trimmed of the eol bytes

Exceptions

FILELINEITERATOR-OPEN-ERROR	the given file cannot be opened for reading (arg will be assigned to the errno value)
ENCODING-CONVERSION-ERROR	this exception could be thrown if the eol argument has a different character encoding from the File's and an error occurs during encoding conversion
ILLEGAL-EXPRESSION	FileLineIterator::constructor() cannot be called with a TTY target when %no-terminal-io is set

45.17.2.2 Qore::FileLineIterator::copy ( )

Creates a new FileLineIterator object, based on the same object being iterated in the original object (the original file is reopened)

Example:

```
my FileLineIterator $ni = $i.copy();
```

Exceptions

FILELINEITERATOR-COPY-ERROR	the original file cannot be reopened for reading (arg will be assigned to the errno value)
ILLEGAL-EXPRESSION	FileLineIterator::constructor() cannot be called with a TTY target when %no-terminal-io is set

45.17.2.3 string Qore::FileLineIterator::getEncoding ( )

Returns the character encoding for the FileLineIterator.

Code Flags:

CONSTANT

Example:

```
my string $encoding = $f.getEncoding();
```

Returns

the character encoding for the FileLineIterator

45.17.2.4 string Qore::FileLineIterator::getFileName ( )

returns the file path/name used to open the file

Code Flags:

CONSTANT

Example:

```
my string $fn = $f.getFileName();
```

Returns

the file path/name used to open the file

45.17.2.5 string Qore::FileLineIterator::getLine ( )

returns the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getLine());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

See also

FileLineIterator::getValue()

45.17.2.6 int Qore::FileLineIterator::getPos ( )

Returns the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $pos = $i.getPos();
```

Returns

the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::setPos()

45.17.2.7 string Qore::FileLineIterator::getValue ( ) [virtual]

returns the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current line in the file or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

See also

FileLineIterator::getLine()

Implements Qore::AbstractIterator.

45.17.2.8 int Qore::FileLineIterator::index ( )

returns the current iterator line number in the file (the first line is line 1) or 0 if not pointing at a valid element

Returns

the current iterator line number in the file (the first line is line 1) or 0 if not pointing at a valid element

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d: %y\n", $i.index(), $i.getValue());
}
```

45.17.2.9 bool Qore::FileLineIterator::isTty ( )

returns True if the FileLineIterator is connected to a terminal device, False if not

Returns

True if the FileLineIterator is connected to a terminal device, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $i.isTty();
```

45.17.2.10 bool Qore::FileLineIterator::next ( ) [virtual]

Moves the current position to the next line in the file; returns False if there are no more lines to read; if the iterator is not pointing at a valid element before this call, the iterator will be positioned to the beginning of the file.

This method will return True again after it returns False once if file is not empty, otherwise it will always return False The iterator object should not be used after this method returns False

Returns

False if there are no more lines in the file (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf("line: %y\n", $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

45.17.2.11 Qore::FileLineIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.17.2.12 bool Qore::FileLineIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.18 Qore::FtpClient Class Reference

The FtpClient class allows Qore code to communicate with FTP servers with the FTP and FTPS protocols.

Public Member Functions

- clearStats ()
  Clears performance statistics.
- nothing clearWarningQueue ()
  Removes any warning Queue object from the Socket.
- nothing connect ()
  Connects to the FTP server and attempts a login; if any errors occur, an exception is thrown.
- constructor ()
  Creates an empty FtpClient object.
- constructor (string url)
  Creates an FtpClient object and initializes it with a URL.
- copy ()
  Throws an exception to prevent copying of objects this class.
- nothing cwd (string path)
  Changes the current working directory on the server.
- nothing del (string remote_path)
  Deletes a file from the FTP server; if any errors occur, an exception is thrown.
- destructor ()
  Disconnects any remote connection and destroys the object.
- nothing disconnect ()
  Disconnects from an FTP server.
- nothing get (string remote_path, __7_ string local_path)
  Gets a file from the FTP server and stores it on the local filesystem; if any errors occur, an exception is thrown.
- binary getAsBinary (string remote_path)
  Gets a file from the FTP server and returns it as a binary.
- string getAsString (string remote_path)
  Gets a file from the FTP server and returns it as a string.
- __7_ string getHostName ()
  Returns the current hostname value or NOTHING if none is set.

- __7_ string getPassword ()

  Returns the current password value or NOTHING if none is set.
- int getPort ()

  Retrieves the current connection port value for this object.
- __7_ string getSSLCipherName ()

  Returns the name of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established.
- __7_ string getSSLCipherVersion ()

  Returns the version of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established.
- string getURL ()

  Retrieves the current connection URL string for this object.
- hash getUsageInfo ()

  Returns performance statistics for the socket.
- __7_ string getUserName ()

  Returns the current username value or NOTHING if none is set.
- bool isDataSecure ()

  Returns True if the data connections are secure TLS/SSL connections, False if not.
- bool isSecure ()

  Returns True if the control connection is a secure TLS/SSL connection, False if not.
- __7_ string list ()

  Returns a list of files from the FTP server in the server's long format in the current working directory or NOTHING if the path cannot be found.
- __7_ string list (string path)

  Returns a list of files from the FTP server in the server's long format for the given path or NOTHING if the path cannot be found.
- nothing mkdir (string remote_path)

  Creates a directory on the FTP server; if any errors occur, an exception is thrown.
- __7_ string nlst ()

  Returns a list of file names from the FTP server in the current working directory or NOTHING if the path cannot be found.
- __7_ string nlst (string path)

  Returns a list of file names from the FTP server for the given path or NOTHING if the path cannot be found.
- nothing put (string local_path, __7_ string remote_path)

  Transfers a local file to the FTP server; if any errors occur, an exception is thrown.
- nothing putData (string data, string remote_path)

  Transfers string data to the FTP server and saves it as a file on the remote machine; if any errors occur, an exception is thrown.
- nothing putData (binary data, string remote_path)

  Transfers binary data to the FTP server and saves it as a file on the remote machine; if any errors occur, an exception is thrown.
- string pwd ()

  Returns the server-side current working directory.
- nothing rename (string from, string to)

  Renames/moves a file or directory; if any errors occur, an exception is thrown.
- nothing rmdir (string remote_path)

  Removes a directory on the remote FTP server; if any errors occur, an exception is thrown.
- nothing setControlEventQueue ()

  Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on the control connection.
- nothing setControlEventQueue (Qore::Thread::Queue queue)

  Sets a Queue object to receive FtpClient and Socket events on the control connection.

- nothing setDataEventQueue ()

  Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on the data connection.
- nothing setDataEventQueue (Qore::Thread::Queue queue)

  Sets a Queue object to receive FtpClient and Socket events on the data connection.
- nothing setEventQueue ()

  Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on either the data or control connections.
- nothing setEventQueue (Qore::Thread::Queue queue)

  Sets a Queue object to receive FtpClient and Socket events on both the data and control connections.
- nothing setHostName (string host)

  Sets the hostname or address to use to connect to for the next connection.
- nothing setInsecure ()

  Make a non-encrypted connection to the server on the next connect.
- nothing setInsecureData ()

  Make a non-encrypted data connection to the server on the next connect even if the control connection is secure.
- nothing setModeAuto ()

  Sets the object to automatically try to negotiate the data connections in `EPSV`, `PASV`, and `PORT` modes, in that order.
- nothing setModeEPSV ()

  Sets the object to only try to make data connections using `EPSV` (`RFC-2428` extended passive) mode.
- nothing setModePASV ()

  Sets the object to only try to make data connections using `PASV` (`RFC-959` passive) mode.
- nothing setModePORT ()

  Sets the object to only try to make data connections using `PORT` mode.
- nothing setPassword (string pass)

  Sets the password to use for the next connection.
- nothing setPort (int port)

  Sets the control port value to use for the next connection (the FTP protocol default is 21)
- nothing setSecure (bool secure=True)

  Make an FTPS connection to the server on the next connect if the argument is True.
- nothing setURL (string url)

  Sets the connection and login parameters based on the URL passed as an argument.
- nothing setUserName (string user)

  Sets the user name to use for the next connection.
- nothing setWarningQueue (int warning_ms, int warning_bs, Queue queue, any arg, timeout min_ms=1s)

  Sets a Queue object to receive socket warnings.
- __7_ string verifyPeerCertificate ()

  Returns a string code giving the result of verifying the remote certificate or NOTHING if an encrypted connection is not established.

45.18.1 Detailed Description

The FtpClient class allows Qore code to communicate with FTP servers with the FTP and FTPS protocols.

Restrictions:

Qore::PO_NO_NETWORK

The constructor takes an optional URL with the following format:
`[(ftp|ftps)://][`*username*`[:`*password*`]@]`*hostname*`[:`*port*`]`

If the URL is not set with the constructor, then the connection parameters must be set with the FtpClient::set∗() methods. At the very minimum the hostname must be set. If any path name is given in the URL, it is ignored. See the following table for default URL parameters.

FtpClient::constructor() Default URL Parameters

Field	Default Value
protocol	"ftp" (unencrypted)
username	"anonymous"
password	"qore@nohost.com"
port	21

Once the URL (at least the hostname) has been set, any method requiring a connection will make an implicit call to FtpClient::connect() to attempt to connect and login to the FTP server before executing the method.

Objects of this class are capable of making encrypted FTPS connections according to RFC-4217. TLS/SSL encrypted control and data connection will be attempted if the protocol is set to "ftps" or the FtpClient::setSecure() method is called before connecting.

Note that "sftp", or FTP over ssh, is not supported with this class; FTPS is an extension of the FTP protocol to allow for secure connections; while "sftp" is FTP over an encrypted ssh connection. For sftp and ssh support, use the ssh2 module.

When a data connection is required, by default the following modes are tried in series:

- EPSV (Extended Passive Mode)
- PASV (Passive Mode)
- PORT (Port mode)

If the FTP server does not support one of these methods, or network conditions do not allow a data connection of any of these types to be established, then an exception is thrown.

To manually control which modes are tried, see the FtpClient::setModeEPSV(), FtpClient::setModePASV(), and FtpClient::setModePORT() methods.

This class supports posting network events to a Queue, either events on the control or data channels or both can be monitored. See I/O Event Handling for more information.

The events raised by this object are listed in the following table:

FtpClient Events

Name	Value	Description
EVENT_FTP_SEND_MESSAGE	9	Raised immediately before an FTP control message is sent
EVENT_FTP_MESSAGE_RECEIVED	10	Raised when an FTP reply is received on the control channel

Note

This class is not available with the PO_NO_NETWORK parse option.

45.18.2 Member Function Documentation

45.18.2.1 Qore::FtpClient::clearStats ( )

Clears performance statistics.

Example:

```
$ftp.clearStats();
```

Since

Qore 0.8.9

See also

FtpClient::getUsageInfo()

45.18.2.2 nothing Qore::FtpClient::clearWarningQueue ( )

Removes any warning Queue object from the Socket.

Example:

```
$ftp.clearWarningQueue();
```

See also

FtpClient::setWarningQueue()

Since

Qore 0.8.9

45.18.2.3 nothing Qore::FtpClient::connect ( )

Connects to the FTP server and attempts a login; if any errors occur, an exception is thrown.

This method does not need to be called explicitly; connections are established implicitly by all methdods that communicate with FTP servers.

Connection parameters are set in the constructor or in the FtpClient::set∗() methods

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED, EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.connect();
```

Exceptions

FTP-RESPONSE-ERROR	Invalid response received from FTP server
SOCKET-CONNECT-ERROR	Cannot establish socket connection to FTP server
FTP-CONNECT-ERROR	no hostname set, FTP server reported an error, etc
FTP-LOGIN-ERROR	Login denied by FTP server

45.18.2.4 Qore::FtpClient::constructor ( )

Creates an empty FtpClient object.

Example:

```
my FtpClient $ftp();
```

45.18.2.5 Qore::FtpClient::constructor (**string** *url*)

Creates an FtpClient object and initializes it with a URL.

It accepts one argument that will set the URL for the FTP connection; the path is ignored in the URL, however the username, password, hostname and port are respected; additionally if the protocol (or scheme) is "ftps", the client will attempt to establish a secure connection to the server according to RFC-4217 when the first connection is established.

Parameters

url	The URL of the server to connect to; must have at least a hostname

Example:

```
my FtpClient $ftp("ftp://user:pass@example.com:2021");
```

Exceptions

UNSUPPORTED-PROTOCOL	Only "ftp" or "ftps" are allowed as the protocol in the URL
FTP-URL-ERROR	No hostname given in the URL

45.18.2.6 Qore::FtpClient::copy ( )

Throws an exception to prevent copying of objects this class.

Exceptions

FTPCLIENT-COPY-ERROR	FtpClient objects cannot be copied

45.18.2.7 nothing Qore::FtpClient::cwd (**string** *path*)

Changes the current working directory on the server.

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.cwd("/foo/bar");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-CWD-ERROR	FTP server returned an error to the CWD command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.8 nothing Qore::FtpClient::del (**string** *remote_path*)

Deletes a file from the FTP server; if any errors occur, an exception is thrown.

Parameters

remote_path	The path on the server to the file to delete

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.del("file-2.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-DELETE-ERROR	FTP server returned an error to the `DELE` command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.9 Qore::FtpClient::destructor ( )

Disconnects any remote connection and destroys the object.

Events:

EVENT_CHANNEL_CLOSED, EVENT_DELETED

Example:

```
delete $ftp;
```

45.18.2.10 nothing Qore::FtpClient::disconnect ( )

Disconnects from an FTP server.

Events:

EVENT_CHANNEL_CLOSED

Example:

```
delete $ftp;
```

45.18.2.11 nothing Qore::FtpClient::get (string *remote_path*, *__7_* string *local_path*)

Gets a file from the FTP server and stores it on the local filesystem; if any errors occur, an exception is thrown.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

remote_path	The path on the server to the file to get
local_path	The optional local path for saving the file; if this parameter is not given in the call, then the server path is used for the local path as well

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.get("file.txt", "/tmp/file-1.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-FILE-OPEN-ERROR	Could not create the local file
FTP-GET-ERROR	There was an error retrieving the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::getAsString(), FtpClient::getAsData()

45.18.2.12 binary Qore::FtpClient::getAsBinary (string *remote_path*)

Gets a file from the FTP server and returns it as a binary.

Parameters

remote_path	The path on the server to the file to get

Returns

The file data retrieved; if any errors occur an exception is raised

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my binary $b = $ftp.getAsBinary("file.pdf");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-FILE-OPEN-ERROR	Could not create the local file
FTP-GET-ERROR	There was an error retrieving the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::getAsString() for a similar function returning the file's contents as a string; see FtpClient::get() for a function that will get a remote file and save it on the local filesystem

45.18.2.13 string Qore::FtpClient::getAsString (string *remote_path*)

Gets a file from the FTP server and returns it as a string.

Parameters

remote_path	The path on the server to the file to get

Returns

The file retrieved as a string; if any errors occur an exception is raised

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my string $str = $ftp.getAsString("file.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-FILE-OPEN-ERROR	Could not create the local file
FTP-GET-ERROR	There was an error retrieving the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::getAsBinary() for a similar function returning the file's contents as a binary; see FtpClient::get() for a function that will get a remote file and save it on the local filesystem

45.18.2.14 __7_ string Qore::FtpClient::getHostName ( )

Returns the current hostname value or NOTHING if none is set.

Returns

the current hostname value that will be used for the next connection or NOTHING if none is set

Example:

```
my *string $host = $ftp.getHostName();
```

45.18.2.15 __7_ string Qore::FtpClient::getPassword ( )

Returns the current password value or NOTHING if none is set.

Returns

the current password value that will be used for the next connection or NOTHING if none is set

Example:

```
my *string $pass = $ftp.getPassword();
```

45.18.2.16 int Qore::FtpClient::getPort ( )

Retrieves the current connection port value for this object.

Returns

the current port value that will be used for the next connection

Example:

```
my int $port = $ftp.getPort();
```

45.18.2.17 __7_ string Qore::FtpClient::getSSLCipherName ( )

Returns the name of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established.

Returns

the name of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established

Code Flags:

CONSTANT

Example:

```
if ($ftp.isSecure())
    printf("secure connection: %s %s\n", $ftp.getSSLCipherName(), $ftp.getSSLCipherVersion());
```

45.18.2.18 __7_ string Qore::FtpClient::getSSLCipherVersion ( )

Returns the version of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established.

Returns

the version of the cipher for an encrypted connection or NOTHING if an encrypted connection is not established

Code Flags:

CONSTANT

Example:

```
if ($ftp.isSecure())
    printf("secure connection: %s %s\n", $ftp.getSSLCipherName(), $ftp.getSSLCipherVersion());
```

45.18.2.19 string Qore::FtpClient::getURL ( )

Retrieves the current connection URL string for this object.

Returns

the current connection URL string for this object

Example:

```
my string $url = $ftp.getURL();
```

45.18.2.20 hash Qore::FtpClient::getUsageInfo ( )

Returns performance statistics for the socket.

Code Flags:

CONSTANT

Example:

```
my hash $h = $ftp.getUsageInfo();
```

Returns

a hash with the following keys:

- `"bytes_sent"`: an integer giving the total amount of bytes sent
- `"bytes_recv"`: an integer giving the total amount of bytes received
- `"us_sent"`: an integer giving the total number of microseconds spent sending data
- `"us_recv"`: an integer giving the total number of microseconds spent receiving data
- `"arg"`: (only if warning values have been set with FtpClient::setWarningQueue()) the optional argument for warning hashes
- `"timeout"`: (only if warning values have been set with FtpClient::setWarningQueue()) the warning timeout in microseconds
- `"min_throughput"`: (only if warning values have been set with FtpClient::setWarningQueue()) the minimum warning throughput in bytes/sec

Since

Qore 0.8.9

See also

FtpClient::clearStats()

45.18.2.21 __7_ string Qore::FtpClient::getUserName ( )

Returns the current username value or NOTHING if none is set.

Returns

the current username value that will be used for the next connection or NOTHING if none is set

Example:

```
my *string $user = $ftp.getUserName();
```

45.18.2.22 bool Qore::FtpClient::isDataSecure ( )

Returns True if the data connections are secure TLS/SSL connections, False if not.

Returns

True if the data connections are secure TLS/SSL connections, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $ftp.isDataSecure();
```

45.18.2.23 bool Qore::FtpClient::isSecure ( )

Returns True if the control connection is a secure TLS/SSL connection, False if not.

Returns

True if the control connection is a secure TLS/SSL connection, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $ftp.isSecure();
```

45.18.2.24 __7_ string Qore::FtpClient::list ( )

Returns a list of files from the FTP server in the server's long format in the current working directory or NOTHING if the path cannot be found.

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Returns

a list of files from the FTP server in the server's long format in the current working directory or NOTHING if the path cannot be found

Example:

```
my *string $str = $ftp.list();
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-LIST-ERROR	FTP server returned an error in response to the LIST command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::nlst()

45.18.2.25 __7_ string Qore::FtpClient::list (string *path*)

Returns a list of files from the FTP server in the server's long format for the given path or NOTHING if the path cannot be found.

Parameters

path	The path or filename (or filename filter) to list (ex: "/tmp/*.txt")

Returns

a list of files from the FTP server in the server's long format for the given path or NOTHING if the path cannot be found

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my *string $str = $ftp.list("/tmp/*.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-LIST-ERROR	FTP server returned an error in response to the LIST command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::nlst()

45.18.2.26 nothing Qore::FtpClient::mkdir (string *remote_path*)

Creates a directory on the FTP server; if any errors occur, an exception is thrown.

Parameters

remote_path	The path on the server to the directory to create

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.mkdir("tmp");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-MKDIR-ERROR	FTP server returned an error to the MKD command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.27 __7_ string Qore::FtpClient::nlst ( )

Returns a list of file names from the FTP server in the current working directory or NOTHING if the path cannot be found.

Returns

a list of file names from the FTP server in the current working directory or NOTHING if the path cannot be found

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my *string $str = $ftp.nlst();
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-LIST-ERROR	FTP server returned an error in response to the NLST command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::list()

45.18.2.28 __7_ string Qore::FtpClient::nlst (string *path*)

Returns a list of file names from the FTP server for the given path or NOTHING if the path cannot be found.

Parameters

path	The path or filename (or filename filter) to list (ex: `"/tmp/*.txt"`)

Returns

a list of file names from the FTP server for the given path or NOTHING if the path cannot be found

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my *string $str = $ftp.nlst("/tmp/*.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-LIST-ERROR	FTP server returned an error in response to the LIST command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::list()

45.18.2.29 nothing Qore::FtpClient::put (**string** *local_path,* __7_ **string** *remote_path*)

Transfers a local file to the FTP server; if any errors occur, an exception is thrown.

Restrictions:

Qore::PO_NO_FILESYSTEM

Parameters

local_path	The path on the local system of the file to send
remote_path	If given, where to save the file on the server (otherwise the local path is used)

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.put("/tmp/file-1.txt", "file.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-FILE-OPEN-ERROR	Could not open local file for reading
FTP-FILE-PUT-ERROR	Could not determine file size of local file (stat() failed)
FTP-PUT-ERROR	An error occurred while sending the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

See also

FtpClient::putData()

45.18.2.30 nothing Qore::FtpClient::putData (**string** *data,* **string** *remote_path*)

Transfers string data to the FTP server and saves it as a file on the remote machine; if any errors occur, an exception is thrown.

Parameters

data	The file data to save on the remote server
remote_path	The path on the remote server for the file to save the data under

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.putData($str, "file.txt");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-PUT-ERROR	An error occurred while sending the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.31 nothing Qore::FtpClient::putData (binary *data*, string *remote_path*)

Transfers binary data to the FTP server and saves it as a file on the remote machine; if any errors occur, an exception is thrown.

Parameters

data	The file data to save on the remote server
remote_path	The path on the remote server for the file to save the data under

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.putData($bin, "file.pdf");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-PUT-ERROR	An error occurred while sending the file

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.32 string Qore::FtpClient::pwd ( )

Returns the server-side current working directory.

Returns

the server-side current working directory

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
my string $str = $ftp.pwd();
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-PWD-ERROR	FTP server returned an error to the PWD command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.33 nothing Qore::FtpClient::rename (string *from*, string *to*)

Renames/moves a file or directory; if any errors occur, an exception is thrown.

Parameters

from	The original file path on the server to rename/move
to	The new path on the server

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.rename("file.txt", "file.txt.orig");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-RENAME-ERROR	FTP server returned an error to the `RNFR` or `RNTO` commands

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.34 nothing Qore::FtpClient::rmdir (string *remote_path*)

Removes a directory on the remote FTP server; if any errors occur, an exception is thrown.

Parameters

remote_path	The path on the server to the directory to delete

Events:

EVENT_FTP_SEND_MESSAGE, EVENT_FTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_PACKET_SENT

Example:

```
$ftp.rmdir("tmp");
```

Exceptions

FTP-RECEIVE-ERROR	Incomplete message received on control port
FTP-RMDIR-ERROR	FTP server returned an error to the RMD command

Note

see FtpClient::connect() for additional exceptions that could be thrown when connections are implicitly established

45.18.2.35 nothing Qore::FtpClient::setControlEventQueue ( )

Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on the control connection.

Example:

```
$ftp.setControlEventQueue();
```

45.18.2.36 nothing Qore::FtpClient::setControlEventQueue (Qore::Thread::Queue *queue*)

Sets a Queue object to receive FtpClient and Socket events on the control connection.

Parameters

queue	a Queue object to receive FtpClient and Socket events on the control connection; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Example:

```
$ftp.setControlEventQueue($queue);
```

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

See also

event_handling for more information

45.18.2.37 nothing Qore::FtpClient::setDataEventQueue ( )

Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on the data connection.

Example:

```
$ftp.setDataEventQueue();
```

45.18.2.38 nothing Qore::FtpClient::setDataEventQueue (Qore::Thread::Queue *queue*)

Sets a Queue object to receive FtpClient and Socket events on the data connection.

Parameters

queue	a Queue object to receive FtpClient and Socket events on the data connection; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Example:

```
$ftp.setDataEventQueue($queue);
```

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

See also

event_handling for more information

45.18.2.39 nothing Qore::FtpClient::setEventQueue ( )

Clears any Queue object that may be set on the FtpClient object so that I/O events are no longer captured on the object on either the data or control connections.

Example:

```
$ftp.setEventQueue();
```

45.18.2.40 nothing Qore::FtpClient::setEventQueue (Qore::Thread::Queue *queue*)

Sets a Queue object to receive FtpClient and Socket events on both the data and control connections.

Parameters

queue	a Queue object to receive FtpClient and Socket events on both the data and control connections; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Example:

```
$ftp.setEventQueue($queue);
```

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

See also

event_handling for more information

45.18.2.41 nothing Qore::FtpClient::setHostName (string *host*)

Sets the hostname or address to use to connect to for the next connection.

Parameters

host	the hostname or address to use to connect to for the next connection

Example:

```
$ftp.setHostName("example.com");
```

45.18.2.42 nothing Qore::FtpClient::setInsecure ( )

Make a non-encrypted connection to the server on the next connect.

Example:

```
$ftp.setInsecure();
```

Exceptions

SET-INSECURE-ERROR	this method cannot be called while the control connection is established

45.18.2.43 nothing Qore::FtpClient::setInsecureData ( )

Make a non-encrypted data connection to the server on the next connect even if the control connection is secure.

Example:

```
$ftp.setSecure();
$ftp.setInsecureData();
```

Exceptions

SET-INSECUREDATA-ERROR	this method cannot be called while the control connection is established

45.18.2.44 nothing Qore::FtpClient::setModeAuto ( )

Sets the object to automatically try to negotiate the data connections in `EPSV`, `PASV`, and `PORT` modes, in that order.

Example:

```
$ftp.setModeAuto();
```

45.18.2.45 nothing Qore::FtpClient::setModeEPSV ( )

Sets the object to only try to make data connections using `EPSV` (`RFC-2428` extended passive) mode.

Example:

```
$ftp.setModeEPSV();
```

45.18.2.46 nothing Qore::FtpClient::setModePASV ( )

Sets the object to only try to make data connections using PASV (RFC-959 passive) mode.

Example:

```
$ftp.setModePASV();
```

45.18.2.47 nothing Qore::FtpClient::setModePORT ( )

Sets the object to only try to make data connections using PORT mode.

Example:

```
$ftp.setModePORT();
```

45.18.2.48 nothing Qore::FtpClient::setPassword (string *pass*)

Sets the password to use for the next connection.

Parameters

pass	the password to use for the next connection

Example:

```
$ftp.setPassword("ftp");
```

45.18.2.49 nothing Qore::FtpClient::setPort (int *port*)

Sets the control port value to use for the next connection (the FTP protocol default is 21)

Parameters

port	the control port value to use for the next connection (the FTP protocol default is 21); must be > 0 or an FTPCLIENT-SETPORT-PARAMETER-ERROR exception is raised

Example:

```
$ftp.setPort(2021);
```

Exceptions

FTPCLIENT-SETPORT-PARAMETER-ERROR	port <= 0 number passed as argument

45.18.2.50 nothing Qore::FtpClient::setSecure (bool *secure* = True)

Make an FTPS connection to the server on the next connect if the argument is True.

Parameters

secure	Make an FTPS connection to the server on the next connect if the argument is True; make a cleartext connection if the argument is False

Example:

```
$ftp.setSecure(True);
```

Exceptions

SET-SECURE-ERROR	this method cannot be called while the control connection is established

45.18.2.51 nothing Qore::FtpClient::setURL (string *url*)

Sets the connection and login parameters based on the URL passed as an argument.

Only "`ftp`" and "`ftps`" protocols (schemes) are accepted; any other protocol (scheme) will cause an exception to be thrown.

Parameters

url	The URL to use for the next connection; the URL must contain at least a hostname

Example:

```
$ftp.setURL("ftps://user:pass@example.com:2021");
```

Exceptions

UNSUPPORTED-PROTOCOL	Only "`ftp`" and "ftps" are allowed as the protocol (scheme) in the URL
FTP-URL-ERROR	No hostname given in the URL

45.18.2.52 nothing Qore::FtpClient::setUserName (string *user*)

Sets the user name to use for the next connection.

Parameters

user	the user name to use for the next connection

Example:

```
$ftp.setUserName("ftp");
```

45.18.2.53 nothing Qore::FtpClient::setWarningQueue (int *warning_ms*, int *warning_bs*, Queue *queue*, any *arg*, timeout *min_ms* = `1s`)

Sets a Queue object to receive socket warnings.

Example:

```
$ftp.setWarningQueue(5000, 5000, $queue, "socket-1");
```

Parameters

warning_ms	the threshold in milliseconds for individual socket actions (send, receive, connect), if exceeded, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-OPERATION-WARNING"` • `"operation"`: a string giving the operation that caused the warning (ex: `"connect"`) • `"us"`: an integer giving the number of microseconds for the operation • `"timeout"`: an integer giving the warning threshold in microseconds • `"arg"`: if any `"arg"` argument is passed to the FtpClient::setWarningQueue() method, it will be included in the warning hash here
warning_bs	value in bytes per second; if any call has performance below this threshold, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-THROUGHPUT-WARNING"` • `"dir"`: either `"send"` or `"recv"` depending on the direction of the data flow • `"bytes"`: the amount of bytes sent • `"us"`: an integer giving the number of microseconds for the operation • `"bytes_sec"`: a float giving the transfer speed in bytes per second • `"threshold"`: an integer giving the warning threshold in bytes per second • `"arg"`: if any `"arg"` argument is passed to the FtpClient::setWarningQueue() method, it will be included in the warning hash here
queue	the Queue object to receive warning events
arg	an optional argument to be placed in the `"arg"` key in each warning hash (could be used to identify the socket for example)
min_ms	the minimum transfer time with a resolution of milliseconds for a transfer to be eligible for triggering a warning; transfers that take less than this period of time are not eligible for raising a warning

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set
SOCKET-SETWARNING-QUEUE-ERROR	at least one of *warning_ms* and *warning_bs* must be > 0

See also

FtpClient::clearWarningQueue()

Since

Qore 0.8.9

45.18.2.54 __7_ string Qore::FtpClient::verifyPeerCertificate ( )

Returns a string code giving the result of verifying the remote certificate or NOTHING if an encrypted connection is not established.

Returns

A string code giving the result of verifying the peer's certificate or NOTHING if a secure connection has not been established. The possible values are found in the keys of the X509_VerificationReasons hash constant. This hash can also be used to generate a textual description of the verification result.

Code Flags:

CONSTANT

Example:

```
if ($ftp.isSecure() && (my *string $str = $ftp.verifyPeerCertificate())) {
    printf("certificate: %s: %s\n", $str, X509_VerificationReasons.$str);
```

45.19 Qore::Thread::Gate Class Reference

The Gate class implements a reentrant thread lock.

Public Member Functions

- constructor ()

  Creates a new Gate object.
- copy ()

  Creates a new Gate object, not based on the original.
- destructor ()

  Destroys the Gate object.
- int enter (timeout timeout_ms)

  Acquires the lock if it is unlocked or locked by the same thread, otherwise blocks until the lock counter reaches zero.
- nothing enter ()

  Increments the lock count if the lock is unlocked or already owned by the same thread, otherwise blocks until the lock counter reaches zero.
- int exit ()

  Decrements the lock counter; if it reaches zero then the lock is unlocked and any blocked threads are awoken; in this case 0 is returned; in all other cases, non-zero is returned.
- int numInside ()

  Returns the current lock count.
- int numWaiting ()

  Returns the number of threads blocked on the Gate.
- int tryEnter ()

  Acquires the lock if it is unlocked or locked by the same thread, in which case this method returns 0, otherwise returns immediately with -1.

45.19.1 Detailed Description

The Gate class implements a reentrant thread lock.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

Once a thread grabs the lock, it can call the Gate::enter() method again without blocking. Other threads that try to enter the lock will block until the thread holding the lock calls Gate::exit() an equal number of times to Gate::enter() calls.

See the AutoGate class for a class that assists in exception-safe Gate locking.

Additionally, the on_exit statement can provide exception-safe Gate handling at the lexical block level as in the following example:

```
{
    $g.enter();
    on_exit
        $g.exit();
    # ... when this block exits the gate lock counter will be decremented,
    #     even in the case of return statements or exceptions
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.19.2 Member Function Documentation

45.19.2.1 Qore::Thread::Gate::constructor ( )

Creates a new Gate object.

Example:

```
my Gate $gate();
```

45.19.2.2 Qore::Thread::Gate::copy ( )

Creates a new Gate object, not based on the original.

Example:

```
my Gate $new_gate = $gate.copy();
```

45.19.2.3 Qore::Thread::Gate::destructor ( )

Destroys the Gate object.

Note that it is a programming error to delete this object while other threads are blocked on it; in this case an exception is thrown in the deleting thread, and in each thread blocked on this object when it is deleted.

Example:

```
delete $gate;
```

Exceptions

LOCK-ERROR	Object deleted while other threads blocked on it

45.19.2.4 int Qore::Thread::Gate::enter (timeout *timeout_ms*)

Acquires the lock if it is unlocked or locked by the same thread, otherwise blocks until the lock counter reaches zero.

Parameters

timeout_ms	a timeout value to wait to acquire the lock (enter the Gate); integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds)

Returns

0 if no timeout occurred, non-zero if a timeout occurred.

Example:

```
if ($gate.enter(1500ms))
    throw "TIMEOUT-ERROR", "gate acquisition timed out after 1.5s";
```

Exceptions

LOCK-ERROR	object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.19.2.5 nothing Qore::Thread::Gate::enter ( )

Increments the lock count if the lock is unlocked or already owned by the same thread, otherwise blocks until the lock counter reaches zero.

Example:

```
$gate.enter();
```

Exceptions

LOCK-ERROR	object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.19.2.6 int Qore::Thread::Gate::exit ( )

Decrements the lock counter; if it reaches zero then the lock is unlocked and any blocked threads are awoken; in this case 0 is returned; in all other cases, non-zero is returned.

Returns

returns 0 if the Gate was unlocked; otherwise returns non-zero

Example:

```
$gate.exit();
```

Exceptions

LOCK-ERROR	lock not owned by the current thread, object deleted in another thread, etc

45.19.2.7 int Qore::Thread::Gate::numInside ( )

Returns the current lock count.

Returns

the current lock count

Code Flags:

CONSTANT

Example:

```
my int $c = $gate.numInside();
```

45.19.2.8 int Qore::Thread::Gate::numWaiting ( )

Returns the number of threads blocked on the Gate.

Code Flags:

CONSTANT

Example:

```
my int $c = $gate.numWaiting();
```

45.19.2.9 int Qore::Thread::Gate::tryEnter ( )

Acquires the lock if it is unlocked or locked by the same thread, in which case this method returns 0, otherwise returns immediately with -1.

Returns

0 for success (acquired the lock) or -1 for failure (would block)

Example:

```
if ($gate.tryEnter()) {
    on_exit $gate.exit();
    printf("YIPEE!  We finally got the gate!\n");
}
```

Exceptions

LOCK-ERROR	object deleted in another thread, etc

45.20 Qore::GetOpt Class Reference

The GetOpt class provides an easy way to process POSIX-style command-line options in Qore scripts/programs.

Public Member Functions

- constructor (hash options)

  Creates the GetOpt object and sets the option hash with the single required argument.
- copy ()

  Throws an exception; objects of this class cannot be copied.
- hash parse (reference pgm_args)

  Parses the parameter list according to the option hash passed to the constructor.

- hash parse (softlist pgm_args)

  Parses the parameter list according to the option hash passed to the constructor.
- hash parse2 (reference pgm_args)

  Parses the parameter list according to the option hash passed to the constructor.
- hash parse2 (softlist pgm_args)

  Parses the parameter list according to the option hash passed to the constructor.
- hash parse3 (reference pgm_args)

  Parses the parameter list according to the option hash passed to the constructor and displays an explanatory error message on stderr and exits the program if an error occurs.
- hash parse3 (softlist pgm_args)

  Parses the parameter list according to the option hash passed to the constructor and displays an explanatory error message on stderr and exits the program if an error occurs.

45.20.1 Detailed Description

The GetOpt class provides an easy way to process POSIX-style command-line options in Qore scripts/programs.

45.20.2 Member Function Documentation

45.20.2.1 Qore::GetOpt::constructor (**hash** *options*)

Creates the GetOpt object and sets the option hash with the single required argument.

Parameters

options	Each key defines the key value for the return hash if any arguments are given corresponding to the string value of the key; The string value of each hash key follows the following pattern: *opts* [(=|**:**) *type* [*modifier*]] Where the meaning of the above placeholders in italics is: • *opts*: At least one short option and/or a long option name; if both are present, then they must be separated by a comma. The short option must be a single character. • (=|**:**) *type*: if **"="** is used, then the option takes a mandatory argument, if **":"** is used, then the argument is optional. Types are specified as follows: – *s*: string – *i*: integer – *f*: float – *d*: date – *b*: boolean • *modifier*: **"@"** specifies a list, **"+"** an additive value (sum; must be integer or float type)

Example:

```
const program_options =
    ( "url"  : "url,u=s",
      "xml"  : "xml,x",
      "lxml" : "literal-xml,X",
      "verb" : "verbose,v",
      "help" : "help,h" );

my GetOpt $getopt(program_options);
```

Exceptions

GETOPT-PARAMETER-ERROR	option key value is not a string; "_ERRORS_" used as option key
GETOPT-OPTION-ERROR	list option specified as optional; empty option key value string; multiple long options given for key; duplicate options given; invalid attributes for option; unknown modifier given for option

45.20.2.2 Qore::GetOpt::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

GETOPT-COPY-ERROR	copying GetOpt objects is not supported

45.20.2.3 hash Qore::GetOpt::parse (reference *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor.

All arguments parsed will be removed from the list reference passed as the sol argument, leaving only unparsed arguments (for example, file names).

If any errors are encountered, the return value hash will have a key "_ERRORS_" giving a list of error messages pertaining to the options parsed.

Parameters

pgm_args	The reference should point to a list of arguments to process (normally $ARGV); any argument accepted by the object will be removed from the list

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument. The hash key "_ERRORS_" will contain any errors.

Example:

```
my hash $o = $getopt.parse(\$ARGV);
if (exists $o."_ERRORS_") {
    foreach my string $err in ($o."_ERRORS_")
        stderr.printf("%s\n", $err);
    exit(1);
}
```

See also

GetOpt::parse2() for a similar method that throws an exception instead of putting error information in the "_ERRORS_" key of the hash value returned

45.20.2.4 hash Qore::GetOpt::parse (softlist *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor.

If any errors are encountered, the return value hash will have a key "_ERRORS_" giving a list of error messages pertaining to the options parsed.

Parameters

pgm_args	A list of arguments to process

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument. The hash key "_ERRORS_" will contain any errors.

Example:

```
my hash $o = $getopt.parse($ARGV);
if (exists $o."_ERRORS_") {
    foreach my string $err in ($o."_ERRORS_")
        stderr.printf("%s\n", $err);
    exit(1);
}
```

See also

GetOpt::parse2() for a similar method that throws an exception instead of putting error information in the "_ERRORS_" key of the hash value returned

45.20.2.5 hash Qore::GetOpt::parse2 (reference *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor.

If any errors are encountered, an appropriate exception will be thrown.

Parameters

pgm_args	The reference should point to a list of arguments to process (normally $ARGV); any argument accepted by the object will be removed from the list

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument

Example:

```
try {
   my hash $o = $getopt.parse2(\$ARGV);
}
catch ($ex) {
   stderr.printf("%s\n", $ex.desc);
   exit(1);
}
```

Exceptions

GETOPT-ERROR	error parsing arguments

See also

GetOpt::parse() for a similar method that puts error information in the "_ERRORS_" key of the hash value returned instead of throwing an exception

45.20.2.6 hash Qore::GetOpt::parse2 (softlist *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor.

If any errors are encountered, an appropriate exception will be thrown.

Parameters

pgm_args	A list of arguments to process

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument

Example:

```
try {
   my hash $o = $getopt.parse2($ARGV);
}
catch ($ex) {
   stderr.printf("%s\n", $ex.desc);
   exit(1);
}
```

Exceptions

GETOPT-ERROR	error parsing arguments

See also

GetOpt::parse() for a similar method that puts error information in the "_ERRORS_" key of the hash value returned instead of throwing an exception

45.20.2.7 hash Qore::GetOpt::parse3 (reference *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor and displays an explanatory error message on stderr and exits the program if an error occurs.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

pgm_args	The reference should point to a list of arguments to process (normally $ARGV); any argument accepted by the object will be removed from the list

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument

Example:

```
my hash $o = $getopt.parse3(\$ARGV);
```

See also

- GetOpt::parse() for a similar method that puts error information in the "_ERRORS_" key of the hash value returned
- GetOpt::parse2() for a similar method that throws exceptions instead of exiting the program

45.20.2.8 hash Qore::GetOpt::parse3 (softlist *pgm_args*)

Parses the parameter list according to the option hash passed to the constructor and displays an explanatory error message on stderr and exits the program if an error occurs.

Restrictions:

Qore::PO_NO_PROCESS_CONTROL

Parameters

pgm_args	A list of arguments to process

Returns

A hash keyed by option names (as given in the hash to the GetOpt constructor), where each key's value is the value of the argument passed in the list argument

Example:

```
my hash $o = $getopt.parse3($ARGV);
```

Exceptions

GETOPT-ERROR	error parsing arguments

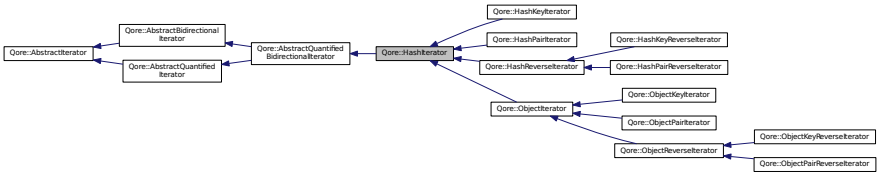
See also

- GetOpt::parse() for a similar method that puts error information in the "_ERRORS_" key of the hash value returned
- GetOpt::parse2() for a similar method that throws exceptions instead of exiting the program

45.21 Qore::HashIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.
- constructor ()

  Creates an empty hash iterator object.
- copy ()

  Creates a copy of the HashIterator object, iterating the same object as the original and in the same position.
- bool empty ()

returns True if the hash is empty; False if not

- bool first ()

  returns True if on the first element of the hash

- string getKey ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

- any getKeyValue ()

  returns the current value of the current has key being iterated or throws an `INVALID-ITERATOR` exception if the iterator is invalid

- any getValue ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

- hash getValuePair ()

  returns a hash with the current key and value (a hash with 2 keys: `"key"` and `"value"`) or throws an `INVALID-ITERATOR` exception if the iterator is invalid

- bool last ()

  returns True if on the last element of the hash

- bool next ()

  Moves the current position to the next element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the hash if the hash is not empty.

- bool prev ()

  Moves the current position to the previous element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the hash if the hash is not empty.

- reset ()

  Reset the iterator instance to its initial state.

- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.21.1 Detailed Description

This class an iterator class for hashes.

Call HashIterator::next() to iterate through the hash; do not use the iterator if HashIterator::next() returns False. A hash can be iterated in reverse order by calling HashIterator::prev() instead of HashIterator::next()

Example: HashIterator basic usage

```
my hash $h = ( "key1" : 1, "key2" : 2, );

my HashIterator $it($h);
while ($it.next()) {
    printf("getKey: %n; getKeyValue: %n; getValue: %n; getValuePair: %n\n",
           $it.getKey(), $it.getKeyValue(), $it.getValue(), $it.getValuePair());
}
```

```
getKey: "key1"; getKeyValue: 1; getValue: 1;
      getValuePair: hash: (key : "key1", value : 1)
getKey: "key2"; getKeyValue: 2; getValue: 2;
      getValuePair: hash: (key : "key2", value : 2)
```

Note

- In general, the HashIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

HashReverseIterator

45.21.2 Member Function Documentation

45.21.2.1 Qore::HashIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashIterator $hi($h);
```

45.21.2.2 Qore::HashIterator::constructor ( )

Creates an empty hash iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashIterator $hi($h);
```

45.21.2.3 Qore::HashIterator::copy ( )

Creates a copy of the HashIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashIterator $ni = $i.copy();
```

45.21.2.4 bool Qore::HashIterator::empty ( ) [virtual]

returns True if the hash is empty; False if not

Returns

True if the hash is empty; False if not

Code Flags:

CONSTANT

Example:

```
if ($i.empty())
    printf("the hash is empty\n");
```

Implements Qore::AbstractQuantifiedIterator.

45.21.2.5 bool Qore::HashIterator::first ( ) [virtual]

returns True if on the first element of the hash

Returns

True if on the first element of the hash

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ObjectReverseIterator, and Qore::HashReverseIterator.

45.21.2.6 string Qore::HashIterator::getKey ( )

returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getKey());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.21.2.7 any Qore::HashIterator::getKeyValue ( )

returns the current value of the current has key being iterated or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current value of the current has key being iterated or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getKeyValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

Qore 0.8.6

45.21.2.8 any Qore::HashIterator::getValue ( ) [virtual]

returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

The current hash key can be returned with getKey().

Returns

the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

Reimplemented in Qore::ObjectPairReverseIterator, Qore::ObjectKeyReverseIterator, Qore::HashKeyReverse↩ Iterator, Qore::HashPairReverseIterator, Qore::ObjectPairIterator, Qore::ObjectKeyIterator, Qore::HashPairIterator, and Qore::HashKeyIterator.

45.21.2.9 hash Qore::HashIterator::getValuePair ( )

returns a hash with the current key and value (a hash with 2 keys: `"key"` and `"value"`) or throws an `INVAL↩ ID-ITERATOR` exception if the iterator is invalid

Returns

a hash with the current key and value (a hash with 2 keys: `"key"` and `"value"`) or throws an `INVALI↩ D-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValuePair());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

Qore 0.8.6

45.21.2.10 bool Qore::HashIterator::last ( ) [virtual]

returns True if on the last element of the hash

Returns

True if on the last element of the hash

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ObjectReverseIterator, and Qore::HashReverseIterator.

45.21.2.11 bool Qore::HashIterator::next ( ) [virtual]

Moves the current position to the next element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the hash if the hash is not empty.

This method will return True again after it returns False once if hash is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the hash (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y = %y\n", $i.getKey(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

Reimplemented in Qore::ObjectReverseIterator, and Qore::HashReverseIterator.

45.21.2.12 bool Qore::HashIterator::prev ( ) [virtual]

Moves the current position to the previous element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the hash if the hash is not empty.

This method will return True again after it returns False once if the hash is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the hash (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + %y = %y\n", $i.getKey(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractBidirectionalIterator.

Reimplemented in Qore::ObjectReverseIterator, and Qore::HashReverseIterator.

45.21.2.13 Qore::HashIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.21.2.14 bool Qore::HashIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

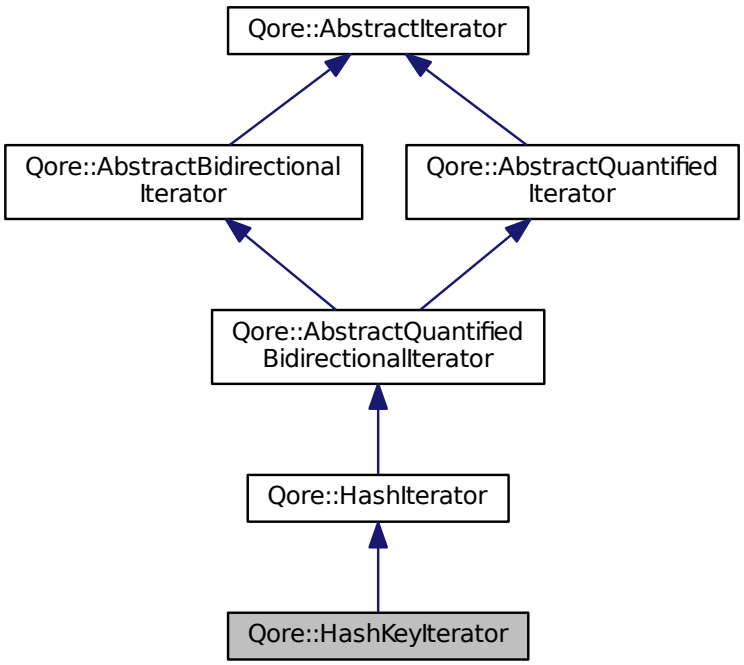
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.22 Qore::HashKeyIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashKeyIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.
- constructor ()

  Creates an empty hash iterator object.
- copy ()

  Creates a copy of the HashKeyIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

45.22.1 Detailed Description

This class an iterator class for hashes.

Call HashKeyIterator::next() to iterate through the hash; do not use the iterator if HashKeyIterator::next() returns False. A hash can be iterated in reverse order by calling HashKeyIterator::prev() instead of HashKeyIterator::next()

Example: HashKeyIterator basic usage

```
my hash $data = (
        "key1" : 1,
```

```
        "key2" : 2,
        "key3" : 3,
    );

my HashKeyIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: "key1"
iter: "key2"
iter: "key3"
```

Note

- In general, the HashKeyIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

HashKeyReverseIterator

45.22.2 Member Function Documentation

45.22.2.1 Qore::HashKeyIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashKeyIterator $hi($h);
```

45.22.2.2 Qore::HashKeyIterator::constructor ( )

Creates an empty hash iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashKeyIterator $hi($h);
```

45.22.2.3 Qore::HashKeyIterator::copy ( )

Creates a copy of the HashKeyIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashKeyIterator $ni = $i.copy();
```

45.22.2.4 string Qore::HashKeyIterator::getValue ( ) [virtual]

returns the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
foreach my string $key in ($hash.keyIterator())
    printf("key: %s\n", $key);
```

Exceptions

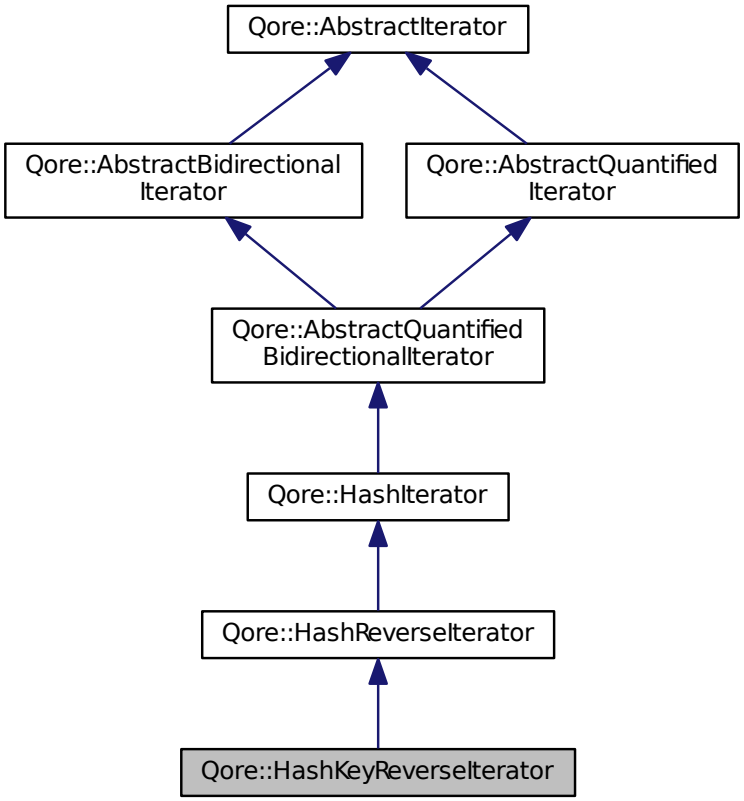
INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.23 Qore::HashKeyReverseIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashKeyReverseIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.
- constructor ()

  Creates an empty iterator object.
- copy ()

  Creates a copy of the HashKeyReverseIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

45.23.1 Detailed Description

This class an iterator class for hashes.

Call HashKeyReverseIterator::next() to iterate through the hash in reverse order; do not use the iterator if HashKeyReverseIterator::next() returns False. A hash can be iterated in reverse order by calling HashKeyReverseIterator::prev() instead of HashKeyReverseIterator::next()

Example: HashKeyReverseIterator basic usage

```
my hash $data = (
        "key1" : 1,
        "key2" : 2,
        "key3" : 3,
    );

my HashKeyReverseIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: "key3"
iter: "key2"
iter: "key1"
```

Note

- In general, the HashKeyReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- HashKeyReverseIterator is functionally equivalent to HashKeyIterator, but the effect of HashKeyReverseIterator::next() and HashKeyReverseIterator::prev() are the opposite of HashKeyIterator::next() and HashKeyIterator::prev(); that is HashKeyReverseIterator::next() will iterate through the hash in reverse order, while HashKeyReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for HashKeyReverseIterator::first() and HashKeyReverseIterator::last() are swapped in respect to HashKeyIterator::first() and HashKeyIterator::last().

See also

HashKeyIterator

45.23.2 Member Function Documentation

45.23.2.1 Qore::HashKeyReverseIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashKeyReverseIterator $hi($h);
```

45.23.2.2 Qore::HashKeyReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashKeyReverseIterator $hi($h);
```

45.23.2.3 Qore::HashKeyReverseIterator::copy ( )

Creates a copy of the HashKeyReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashKeyReverseIterator $ni = $i.copy();
```

45.23.2.4 string Qore::HashKeyReverseIterator::getValue ( ) [virtual]

returns the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
my HashKeyReverseIterator $hi($hash);
while ($hi.next())
    printf("key: %s\n", $hi.getValue());
```

Exceptions

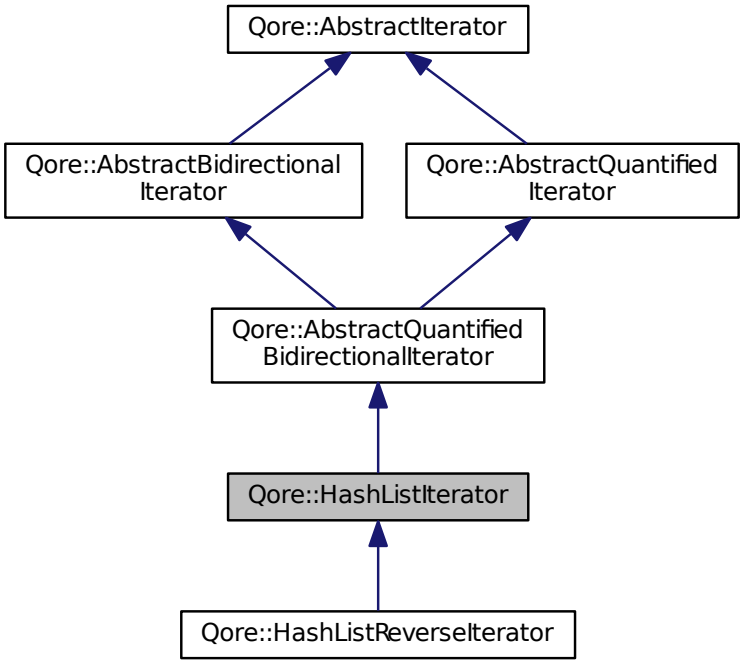
INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.24 Qore::HashListIterator Class Reference

This class an iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

Inheritance diagram for Qore::HashListIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash list iterator object.
- constructor ()

  Creates an empty hash list iterator object.
- copy ()

  Creates a copy of the HashListIterator object, iterating the same object as the original and in the same position.
- bool empty ()

  returns True if the result list is empty; False if not
- bool first ()

  returns True if on the first element of the list
- any getKeyValue (string key)

  Returns the current value for the column given as an argument.
- hash getRow ()

  returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- hash getValue ()

  returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- int index ()

  returns the current iterator position in the list or -1 if not pointing at a valid element
- bool last ()

  returns True if on the last element of the list
- int max ()

  returns the number of elements in the list

- any memberGate (string key)

  This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.
- bool next ()

  Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.
- bool prev ()

  Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.
- reset ()

  Reset the iterator instance to its initial state.
- bool set (int pos)

  sets the new position in the result list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True
- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.24.1 Detailed Description

This class an iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

This class can be used as a more flexible alternative to the context statement.

Call HashListIterator::next() to iterate through the lists of column values assigned to each hash key; do not use the iterator if HashListIterator::next() returns False. A result list can be iterated in reverse order by calling HashListIterator::prev() instead of HashListIterator::next()

Example: HashListIterator basic usge

```
my hash $data = ( "column1" : ( 1, 2, 3, ),
                  "column2" : ( "a", "b", "c", ) );

my HashListIterator $it($data);
while ($it.next()) {
    printf("iter %d: getValue: %n; getKeyValue('column1'): %n\n",
           $it.index(), $it.getValue(), $it.getKeyValue('column1'));
}
```

```
iter 0: getValue: hash: (column1 : 1, column2 : "a"); getKeyValue('column1'): 1
iter 1: getValue: hash: (column1 : 2, column2 : "b"); getKeyValue('column1'): 2
iter 2: getValue: hash: (column1 : 3, column2 : "c"); getKeyValue('column1'): 3
```

Note

- In general, the HashListIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

HashListReverseIterator

45.24.2 Member Function Documentation

45.24.2.1 Qore::HashListIterator::constructor (hash *h*)

Creates the hash list iterator object.

Parameters

h	the hash of lists to iterate

Example:

```
my HashListIterator $i($h);
```

45.24.2.2 Qore::HashListIterator::constructor ( )

Creates an empty hash list iterator object.

Example:

```
my *hash $q = $ds.select("select * from some_table");
my HashListIterator $i($q);
```

45.24.2.3 Qore::HashListIterator::copy ( )

Creates a copy of the HashListIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashListIterator $ni = $i.copy();
```

45.24.2.4 bool Qore::HashListIterator::empty ( ) [virtual]

returns True if the result list is empty; False if not

Returns

True if the result list is empty; False if not

Code Flags:

CONSTANT

Example:

```
if ($i.empty())
    printf("the result list is empty\n");
```

Implements Qore::AbstractQuantifiedIterator.

45.24.2.5 bool Qore::HashListIterator::first ( ) [virtual]

returns True if on the first element of the list

Returns

True if on the first element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::HashListReverseIterator.

45.24.2.6 any Qore::HashListIterator::getKeyValue (string *key*)

Returns the current value for the column given as an argument.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.getKeyValue("value"));
}
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element
HASHLISTITERATOR-ERROR	the hash key given has a value that is not a list

Note

HashListIterator::memberGate() allows for the iterator to act as if it is a hash with members equal to the current row being iterated

45.24.2.7 hash Qore::HashListIterator::getRow ( )

returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Note

equivalent to HashListIterator::getValue()

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element
HASHLISTITERATOR-ERROR	the one of the hash keys has a value that is not a list

45.24.2.8 hash Qore::HashListIterator::getValue ( ) [virtual]

returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Note

- equivalent to HashListIterator::getRow()

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element
HASHLISTITERATOR-ERROR	the one of the hash keys has a value that is not a list

Implements Qore::AbstractIterator.

45.24.2.9 int Qore::HashListIterator::index ( )

returns the current iterator position in the list or -1 if not pointing at a valid element

Returns

the current iterator position in the list or -1 if not pointing at a valid element

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.24.2.10 bool Qore::HashListIterator::last ( ) [virtual]

returns True if on the last element of the list

Returns

True if on the last element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::HashListReverseIterator.

45.24.2.11 int Qore::HashListIterator::max ( )

returns the number of elements in the list

Returns

the number of elements in the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.24.2.12 any Qore::HashListIterator::memberGate (string *key*)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.value);
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element
HASHLISTITERATOR-E↩ RROR	the hash key given has a value that is not a list

Note

equivalent to HashListIterator::getKeyValue() when called explicitly

45.24.2.13 bool Qore::HashListIterator::next ( ) [virtual]

Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

Reimplemented in Qore::HashListReverseIterator.

45.24.2.14 bool Qore::HashListIterator::prev ( ) [virtual]

Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractBidirectionalIterator.

Reimplemented in Qore::HashListReverseIterator.

45.24.2.15 Qore::HashListIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.24.2.16 bool Qore::HashListIterator::set (int *pos*)

sets the new position in the result list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True

Parameters

pos	the new position for the iterator with 0 as the first element

Returns

False, meaning the iterator is not valid, otherwise it returns True

Example:

```
if (!$i.set($pos))
    throw "INVALID-POSITION", sprintf("%d is an invalid position", $pos);
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.24.2.17 bool Qore::HashListIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

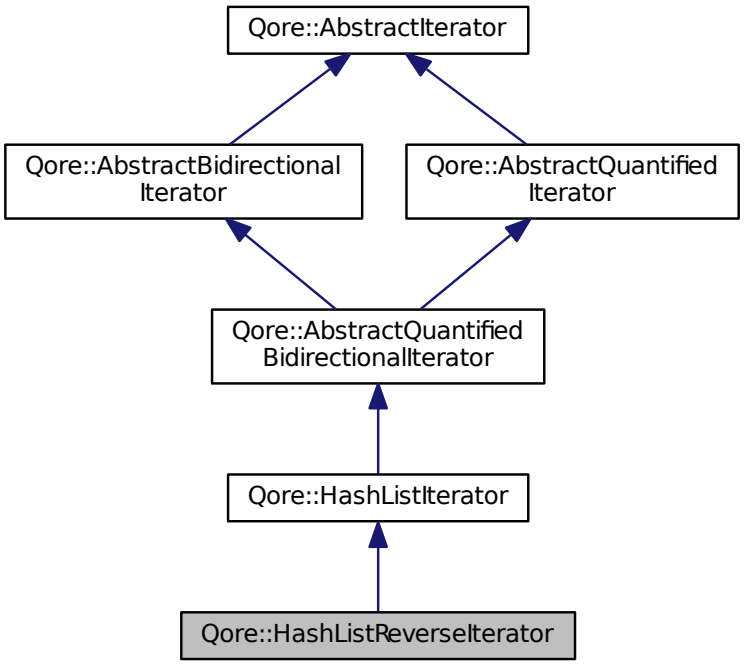
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.25 Qore::HashListReverseIterator Class Reference

This class a reverse iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

Inheritance diagram for Qore::HashListReverseIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash list iterator object.
- constructor ()

  Creates an empty hash list iterator object.
- copy ()

  Creates a copy of the HashListReverseIterator object, iterating the same object as the original and in the same position.
- bool first ()

  returns True if on the first element being iterated in the list (ie the last element in the list)
- bool last ()

returns True if on the last element being iterated in the list (ie the first element in the list)

- any memberGate (string key)

  This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

- bool next ()

  Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

- bool prev ()

  Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

45.25.1 Detailed Description

This class a reverse iterator class for hashes of lists as returned by Qore::SQL::Datasource::select() and Qore::SQL::DatasourcePool::select(), both of which return hashes with keys giving column names where the key values are lists of column values.

Like the Qore::HashListIterator class, this class can be used as a more flexible alternative to the context statement, except this class will iterate the result list in reverse order.

Call HashListReverseIterator::next() to iterate through the lists of column values assigned to each hash key in reverse order; do not use the iterator if HashListReverseIterator::next() returns False. A result list can be iterated in reverse order by calling HashListReverseIterator::prev() instead of HashListReverseIterator::next()

Example: HashListReverseIterator basic usge

```
my hash $data = ( "column1" : ( 1, 2, 3, ),
                  "column2" : ( "a", "b", "c", ) );

my HashListReverseIterator $it($data);
while ($it.next()) {
    printf("iter %d: getValue: %n; getKeyValue('column1'): %n\n",
           $it.index(), $it.getValue(), $it.getKeyValue('column1'));
}

iter 2: getValue: hash: (column1 : 3, column2 : "c"); getKeyValue('column1'): 3
iter 1: getValue: hash: (column1 : 2, column2 : "b"); getKeyValue('column1'): 2
iter 0: getValue: hash: (column1 : 1, column2 : "a"); getKeyValue('column1'): 1
```

Note

- In general, the HashListReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- HashListReverseIterator is functionally equivalent to HashListIterator, but the effect of HashListReverseIterator::next() and HashListReverseIterator::prev() are the opposite of HashListIterator::next() and HashListIterator::prev(); that is HashListReverseIterator::next() will iterate through the hash in reverse order, while HashListReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for HashListReverseIterator::first() and HashListReverseIterator::last() are swapped in respect to HashListIterator::first() and HashListIterator::last().

See also

HashListIterator

45.25.2 Member Function Documentation

45.25.2.1 Qore::HashListReverseIterator::constructor (hash *h*)

Creates the hash list iterator object.

Parameters

h	the hash of lists to iterate

Example:

```
my HashListReverseIterator $i($h);
```

45.25.2.2 Qore::HashListReverseIterator::constructor ( )

Creates an empty hash list iterator object.

Example:

```
my *hash $q = $ds.select("select * from some_table");
my HashListReverseIterator $i($q);
```

45.25.2.3 Qore::HashListReverseIterator::copy ( )

Creates a copy of the HashListReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashListReverseIterator $ni = $i.copy();
```

45.25.2.4 bool Qore::HashListReverseIterator::first ( ) [virtual]

returns True if on the first element being iterated in the list (ie the last element in the list)

Returns

True if on the first element being iterated in the list (ie the last element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Reimplemented from Qore::HashListIterator.

45.25.2.5 bool Qore::HashListReverseIterator::last ( ) [virtual]

returns True if on the last element being iterated in the list (ie the first element in the list)

Returns

True if on the last element being iterated in the list (ie the first element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Reimplemented from Qore::HashListIterator.

45.25.2.6 any Qore::HashListReverseIterator::memberGate (string *key*)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.value);
}
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element
HASHLISTITERATOR-ERROR	the hash key given has a value that is not a list

Note

equivalent to HashListIterator::getKeyValue() when called explicitly

45.25.2.7 bool Qore::HashListReverseIterator::next ( ) [virtual]

Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashListIterator.

45.25.2.8 bool Qore::HashListReverseIterator::prev ( ) [virtual]

Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

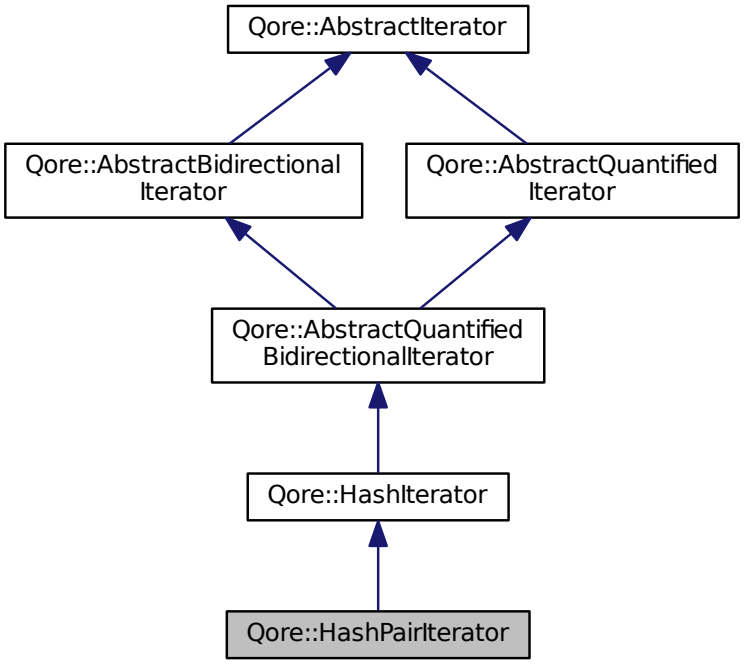
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashListIterator.

45.26 Qore::HashPairIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashPairIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.
- constructor ()

  Creates an empty hash iterator object.
- copy ()

  Creates a copy of the HashPairIterator object, iterating the same object as the original and in the same position.
- hash getValue ()

  returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

45.26.1 Detailed Description

This class an iterator class for hashes.

Call HashPairIterator::next() to iterate through the hash; do not use the iterator if HashPairIterator::next() returns False. A hash can be iterated in reverse order by calling HashPairIterator::prev() instead of HashPairIterator::next()

Example: HashPairIterator basic usage

```
my hash $data = (
        "key1" : 1,
        "key2" : 2,
        "key3" : 3,
    );
```

```
my HashPairIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: hash: (key : "key1", value : 1)
iter: hash: (key : "key2", value : 2)
iter: hash: (key : "key3", value : 3)
```

Note

- In general, the HashPairIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

HashPairReverseIterator

45.26.2 Member Function Documentation

45.26.2.1 Qore::HashPairIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashPairIterator $hi($h);
```

45.26.2.2 Qore::HashPairIterator::constructor ( )

Creates an empty hash iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashPairIterator $hi($h);
```

45.26.2.3 Qore::HashPairIterator::copy ( )

Creates a copy of the HashPairIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashPairIterator $ni = $i.copy();
```

45.26.2.4 hash Qore::HashPairIterator::getValue ( ) [virtual]

returns a hash with the current key and value (a hash with 2 keys: `"key"` and `"value"`) or throws an `INVAL↩ID-ITERATOR` exception if the iterator is invalid

Returns

a hash with the current key and value (a hash with 2 keys: `"key"` and `"value"`) or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
map printf("%s: %y\n", $1.key, $1.value), $hash.pairIterator();
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

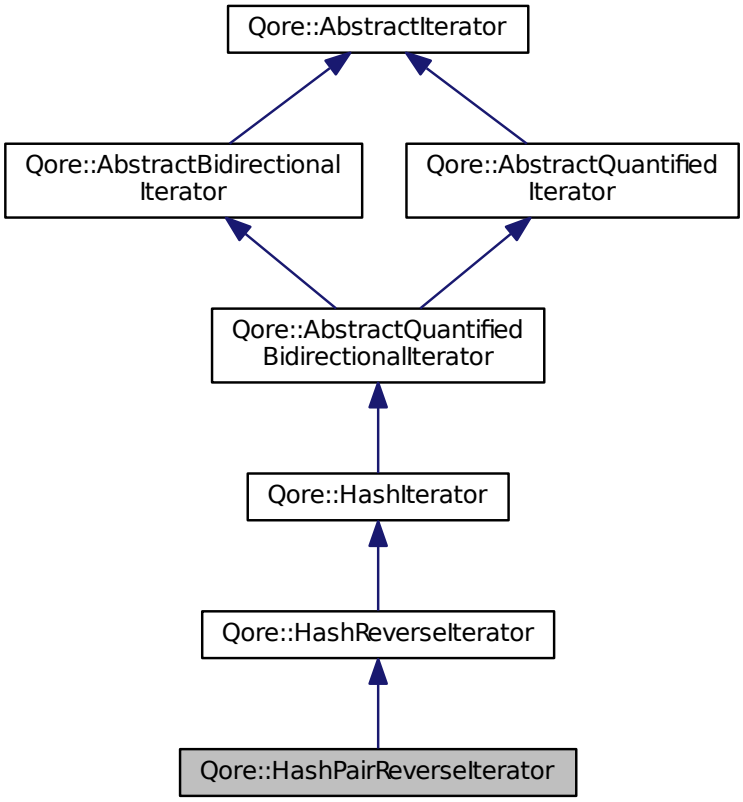
Qore 0.8.6.2

Reimplemented from Qore::HashIterator.

45.27 Qore::HashPairReverseIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashPairReverseIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.
- constructor ()

  Creates an empty iterator object.
- copy ()

  Creates a copy of the HashPairReverseIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

45.27.1 Detailed Description

This class an iterator class for hashes.

Call HashPairReverseIterator::next() to iterate through the hash in reverse order; do not use the iterator if HashPairReverseIterator::next() returns False. A hash can be iterated in reverse order by calling HashPairReverseIterator::prev() instead of HashPairReverseIterator::next()

Example: HashPairReverseIterator basic usage

```
my hash $data = (
        "key1" : 1,
        "key2" : 2,
        "key3" : 3,
    );

my HashPairReverseIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: hash: (key : "key3", value : 3)
iter: hash: (key : "key2", value : 2)
iter: hash: (key : "key1", value : 1)
```

Note

- In general, the HashPairReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- HashPairReverseIterator is functionally equivalent to HashPairIterator, but the effect of HashPairReverseIterator::next() and HashPairReverseIterator::prev() are the opposite of HashPairIterator::next() and HashPairIterator::prev(); that is HashPairReverseIterator::next() will iterate through the hash in reverse order, while HashPairReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for HashPairReverseIterator::first() and HashPairReverseIterator::last() are swapped in respect to HashPairIterator::first() and HashPairIterator::last().

See also

HashPairIterator

45.27.2 Member Function Documentation

45.27.2.1 Qore::HashPairReverseIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashPairReverseIterator $hi($h);
```

45.27.2.2 Qore::HashPairReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashPairReverseIterator $hi($h);
```

45.27.2.3 Qore::HashPairReverseIterator::copy ( )

Creates a copy of the HashPairReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashPairReverseIterator $ni = $i.copy();
```

45.27.2.4 string Qore::HashPairReverseIterator::getValue ( ) [virtual]

returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
map printf("%s: %y\n", $1.key, $1.value), $hash.pairIterator();
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

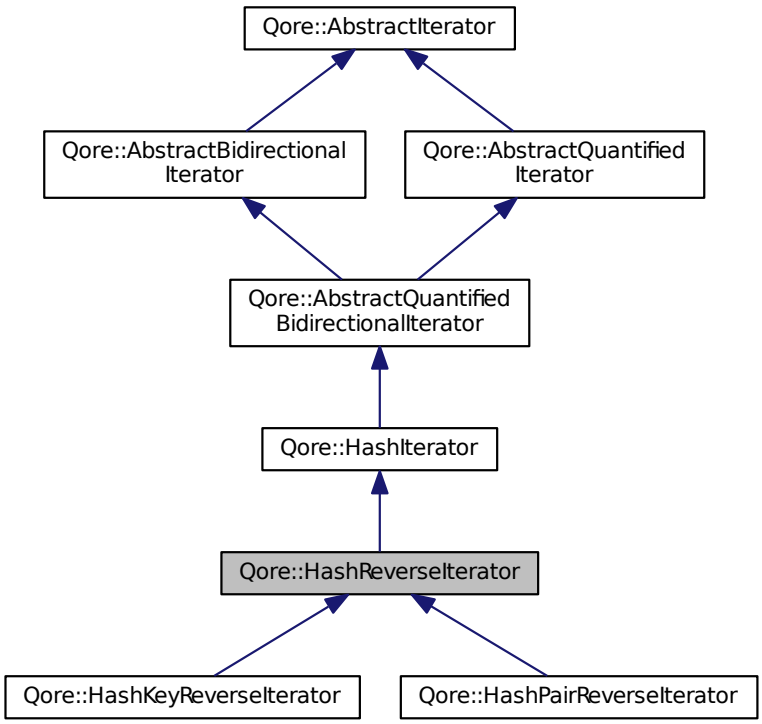
Qore 0.8.6.2

Reimplemented from Qore::HashIterator.

45.28 Qore::HashReverseIterator Class Reference

This class an iterator class for hashes.

Inheritance diagram for Qore::HashReverseIterator:



Public Member Functions

- constructor (hash h)

  Creates the hash iterator object.

- constructor ()

  Creates an empty iterator object.

- copy ()

  Creates a copy of the HashReverseIterator object, iterating the same object as the original and in the same position.

- bool first ()

  returns True if on the last element of the hash

- bool last ()

  returns True if on the first element of the hash

- bool next ()

  Moves the current position to the previous element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the hash if the hash is not empty.

- bool prev ()

  Moves the current position to the next element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the hash if the hash is not empty.

45.28.1 Detailed Description

This class an iterator class for hashes.

Call HashReverseIterator::next() to iterate through the hash in reverse order; do not use the iterator if HashReverseIterator::next() returns False. A hash can be iterated in reverse order by calling HashReverseIterator::prev() instead of HashReverseIterator::next()

Example: HashReverseIterator basic usage

```
my hash $h = ( "key1" : 1, "key2" : 2, );

my HashReverseIterator $it($h);
while ($it.next()) {
    printf("getKey: %n; getKeyValue: %n; getValue: %n; getValuePair: %n\n",
           $it.getKey(), $it.getKeyValue(), $it.getValue(), $it.getValuePair());
}

getKey: "key2"; getKeyValue: 2; getValue: 2;
      getValuePair: hash: (key : "key2", value : 2)
getKey: "key1"; getKeyValue: 1; getValue: 1;
      getValuePair: hash: (key : "key1", value : 1)
```

Note

- In general, the HashReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- HashReverseIterator is functionally equivalent to HashIterator, but the effect of HashReverseIterator::next() and HashReverseIterator::prev() are the opposite of HashIterator::next() and HashIterator::prev(); that is HashReverseIterator::next() will iterate through the hash in reverse order, while HashReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for HashReverseIterator::first() and HashReverseIterator::last() are swapped in respect to HashIterator::first() and HashIterator::last().

See also

HashIterator

45.28.2 Member Function Documentation

45.28.2.1 Qore::HashReverseIterator::constructor (hash *h*)

Creates the hash iterator object.

Parameters

h	the hash to iterate

Example:

```
my HashReverseIterator $hi($h);
```

45.28.2.2 Qore::HashReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *hash $h = get_hash_or_nothing();
my HashReverseIterator $hi($h);
```

45.28.2.3 Qore::HashReverseIterator::copy ( )

Creates a copy of the HashReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my HashReverseIterator $ni = $i.copy();
```

45.28.2.4 bool Qore::HashReverseIterator::first ( ) [virtual]

returns True if on the last element of the hash

Returns

True if on the last element of the hash

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Reimplemented from Qore::HashIterator.

45.28.2.5 bool Qore::HashReverseIterator::last ( ) [virtual]

returns True if on the first element of the hash

Returns

True if on the first element of the hash

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Reimplemented from Qore::HashIterator.

45.28.2.6 bool Qore::HashReverseIterator::next ( ) [virtual]

Moves the current position to the previous element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the hash if the hash is not empty.

This method will return True again after it returns False once if the hash is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the hash (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

HashReverseIterator::next() is the opposite of HashIterator::next(); it is functionally equivalent to HashIterator::prev(); HashReverseIterator::next() iterates through the hash in reverse order

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.28.2.7 bool Qore::HashReverseIterator::prev ( ) [virtual]

Moves the current position to the next element in the hash; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the hash if the hash is not empty.

This method will return True again after it returns False once if hash is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the hash (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

HashReverseIterator::prev() is the opposite of HashIterator::prev(); it is functionally equivalent to HashIterator::next(); HashReverseIterator::prev() iterates through the hash in forward order

Exceptions

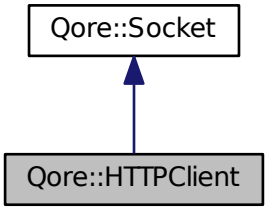
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.29 Qore::HTTPClient Class Reference

The HTTPClient class can be used to communicate with HTTP servers with and without TLS/SSL encryption.

Inheritance diagram for Qore::HTTPClient:



Public Member Functions

- nothing clearProxyURL ()

  Clears the new proxy URL value for the next connection.
- nothing clearProxyUserPassword ()

  Clears the username and password for the next proxy connection.
- clearStats ()

  Clears performance statistics.
- nothing clearUserPassword ()

  Clears the username and password for the connection.
- nothing clearWarningQueue ()

  Removes any warning Queue object from the Socket.
- nothing connect ()

  Connects to the remote socket; SSL/TLS negotiation is performed if required.
- constructor (hash opts)

  Creates the HTTPClient object based on the option parameter passed.
- constructor ()

  Creates the HTTPClient object.
- copy ()

  Copying objects of this class is not supported, an exception will be thrown.
- destructor ()

  Destroys the HTTPClient object and closes any open connections.
- nothing disconnect ()

  Disconnects from the remote socket if a connection is established (otherwise does nothing)
- __7_ string get (string path, __7_ hash headers, __7_ reference info)

  Sends an HTTP `GET` request and returns the message body received as a string or NOTHING if no message body is received.
- int getConnectTimeout ()

  Returns the connect timeout as an integer in milliseconds.
- __7_ string getConnectionPath ()

  Returns the current connection path set in the URL.
- string getDefaultPath ()

  Returns the default path used by the object if no path is set in the URL.
- string getEncoding ()

  Returns the character encoding used for the object.
- string getHTTPVersion ()

Returns the HTTP protocol version string used in outgoing messages.

- int getMaxRedirects ()

  Returns the current `max_redirects` value for the object (the maximum number of HTTP redirects that will be processed before an exception is raised)

- bool getNoDelay ()

  Returns the `TCP_NODELAY` setting for the HTTPClient object.

- __7_ string getProxyURL ()

  Returns the current proxy URL as a string or NOTHING if no proxy URL is set.

- int getTimeout ()

  Returns the default I/O timeout as an integer in milliseconds.

- __7_ string getURL ()

  Returns the current URL.

- hash getUsageInfo ()

  Returns performance statistics for the socket.

- hash head (string path, __7_ hash headers, __7_ reference info)

  Sends an HTTP `HEAD` request and returns as hash of the headers received.

- bool isConnected ()

  Returns True or False giving the current connection state.

- bool isProxySecure ()

  Returns the SSL/TLS flag for the next proxy connection.

- bool isSecure ()

  Returns True if the current connection is encrypted, False if not.

- __7_ string post (string path, string body, __7_ hash headers, __7_ reference info)

  Sends an HTTP `POST` request with a message body and returns the message body received as a string or NOTHING if no message body is received.

- __7_ string post (string path, __7_ binary body, __7_ hash headers, __7_ reference info)

  Sends an HTTP `POST` request with a message body and returns the message body received as a string or NOTHING if no message body is received.

- hash send (string body, string method, __7_ string path, __7_ hash headers, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and optional message body and returns headers and any body received as a response in a hash format.

- hash send (__7_ binary body, string method, __7_ string path, __7_ hash headers, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and optional message body and returns headers and any body received as a response in a hash format.

- nothing sendWithCallbacks (code scb, code rcb, string method, __7_ string path, __7_ hash headers, timeout timeout_ms=0, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and chunked message body as given by a send callback; headers and any body received are returned through a receive callback.

- nothing sendWithRecvCallback (code rcb, string body, string method, __7_ string path, __7_ hash headers, timeout timeout_ms=0, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and optional message body; headers and any body received are returned through a receive callback.

- nothing sendWithRecvCallback (code rcb, __7_ binary body, string method, __7_ string path, __7_ hash headers, timeout timeout_ms=0, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and optional message body; headers and any body received are returned through a receive callback.

- hash sendWithSendCallback (code scb, string method, __7_ string path, __7_ hash headers, timeout timeout_ms=0, softbool getbody=False, __7_ reference info)

  Sends an HTTP request with the specified method and chunked message body as given by a send callback and returns headers and any body received as a response in a hash format.

- nothing setConnectTimeout (timeout timeout_ms=-1)

Sets the connect timeout in milliseconds.

- nothing setDefaultPath (__7_ string path)

  Sets the default path used by the object if no path is set in the URL.

- nothing setEncoding (string encoding)

  Sets the string encoding for the object; any strings deserialized with this object will be tagged with this character encoding.

- nothing setEventQueue ()

  Clears any Queue object that may be set on the HTTPClient object so that I/O events are no longer captured on the object.

- nothing setEventQueue (Qore::Thread::Queue queue)

  Sets a Queue object to receive HTTPClient and Socket events.

- nothing setHTTPVersion (string ver)

  Sets the HTTP protocol version string for headers in outgoing messages, allowed values are `"1.0"` and "1.1".

- nothing setMaxRedirects (softint mr=0)

  Updates the setting for the `max_redirects` value for the object (maximum number of HTTP redirects that will be processed before an exception is raised)

- int setNoDelay (softbool b=True)

  Sets the `TCP_NODELAY` setting for the object.

- setPersistent ()

  temporarily disables implicit reconnections; must be called when the server is already connected

- nothing setProxySecure (softbool b=True)

  Sets the SSL/TLS flag for the next connection to the proxy.

- nothing setProxyURL ()

  Clears the new proxy URL value for the next connection.

- nothing setProxyURL (string url)

  Sets a new proxy URL value for the next connection.

- nothing setProxyUserPassword (string user, string pass)

  Sets the username and password for the connection to the proxy; call after HTTPClient::setProxyURL()

- nothing setProxyUserPassword ()

  Clears the username and password for the next proxy connection.

- nothing setSecure (softbool secure=True)

  Sets the object to make a secure SSL/TLS connection on the next connect if the passed argument is True, or an unencrypted cleartext connection if it is False.

- nothing setTimeout (timeout timeout_ms=0)

  Sets the default I/O timeout value in milliseconds.

- setURL (string url)

  Sets a new URL value for the next connection.

- nothing setUserPassword (string user, string pass)

  Sets the username and password for the connection; call after HTTPClient::setURL()

- nothing setUserPassword ()

  Clears the username and password for the connection.

- nothing setWarningQueue (int warning_ms, int warning_bs, Queue queue, any arg, timeout min_ms=1s)

  Sets a Queue object to receive socket warnings.

45.29.1 Detailed Description

The HTTPClient class can be used to communicate with HTTP servers with and without TLS/SSL encryption.

Restrictions:

Qore::PO_NO_NETWORK

The HTTPClient class can be used to communicate with HTTP servers using the HTTP or HTTPS (HTTP using an SSL/TLS encrypted connection) protocol.

By default `"Connection: Keep-Alive"` is always sent regardless of the HTTP protocol level set for the object, however if a server response contains `"Connection: close"`, the connection will be closed as soon as the full response (including any message body if present) has been read.

HTTP redirect responses are supported and can be limited with the `max_redirects` constructor hash key or by using the HTTPClient::setMaxRedirects() method. The default maximum number of redirects allowed is 5.

HTTP basic authentication is supported; set the username and password in the URL (ex: `"http://username:password@host:port/path"`). To change the URL from the one set by the constructor, call HTTPClient::setURL().

HTTP proxies and basic proxy authentication are supported by setting the proxy constructor hash key to the proxy URL (with a proxy username and password if required) or by calling the HTTPClient::setProxyURL() method.

Objects of this class are thread-safe and support serializing multiple simultaneous requests from many threads. If a request is in progress and another thread attempts to make a request at the same time, the second thread will block until the first is complete. Therefore the total amount of time a thread could wait for a response in a multi-threaded context could be greater than the timeout value (which applies to the maximum time a single internal send() or recv() action can take).

This class understands and automatically decodes `"deflate"`, `"gzip"`, and `"bzip2"` content encodings as well.

The default I/O timeout value is 300,000 milliseconds (5 minutes). Note that the timeout value applies to individual internal send() or recv() operations; for this reason for large transfers the overall I/O time could exceed the timeout value.

When an exception is thrown (for example, a response code of `< 100` or `>= 400` is received from the server), any message body returned will be in the `"arg"` key of the exception hash.

This class understands the protocols in the following table.

HTTPClient Class Protocols

Protocol/Scheme	Default Port	SSL?	Description
`http`	`80`	No	Unencrypted HTTP protocol
`https`	`443`	Yes	HTTP protocol with SSL/TLS encryption

Whenever using an HTTPClient method where a hash of headers can be passed to the method, some headers are generated by default by the class and can be overridden, and some are cannot be overridden and are ignored if passed by the client. See the following tables for details.

HTTPClient Mandatory Headers

Header	Description
`Content-Length`	This header is only sent if a message body is sent, and, if so, the length is calculated automatically.

HTTPClient Default, but Overridable Headers

Header	Default Value
`Accept`	`"text/html"`
`Content-Type`	`"text/html"`
`User-Agent`	`"Qore-HTTP-Client/0.8.8"`
`Connection`	`"Keep-Alive"`
`Accept-Encoding`	`"deflate,gzip,bzip2"`

This class supports posting network events to a Queue. See I/O Event Handling for more information.

The events raised by this object are listed in the following table: **HTTPClient Events**

Name	Description
EVENT_HTTP_CONTENT_LENGTH	Raised when the HTTP "Content-Length" header is received.
EVENT_HTTP_CHUNKED_START	Raised when HTTP chunked data is about to be received.
EVENT_HTTP_CHUNKED_END	Raised when all HTTP chunked data has been received.
EVENT_HTTP_REDIRECT	Raised when an HTTP redirect message is received.
EVENT_HTTP_SEND_MESSAGE	Raised when an HTTP message is sent.
EVENT_HTTP_MESSAGE_RECEIVED	Raised when an HTTP message is received.
EVENT_HTTP_FOOTERS_RECEIVED	Raised when HTTP footers are received.
EVENT_HTTP_CHUNKED_DATA_RECEIVED	Raised when a block of HTTP chunked data is received.
EVENT_HTTP_CHUNK_SIZE	Raised when the next chunk size for HTTP chunked data is known.

Note

- This class is not available with the PO_NO_NETWORK parse option.
- URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme:///socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: `"http://socket=%2ftmp%socket-dir%2fsocket-file`; this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

45.29.2 Member Function Documentation

45.29.2.1 nothing Qore::HTTPClient::clearProxyURL ( )

Clears the new proxy URL value for the next connection.

Example:

```
$httpclient.setProxyURL();
```

45.29.2.2 nothing Qore::HTTPClient::clearProxyUserPassword ( )

Clears the username and password for the next proxy connection.

Call this method after calling HTTPClient::setProxyURL() to clear any proxy authentication information present in the URL used in HTTPClient::setProxyURL()

Example:

```
$httpclient.clearProxyUserPassword();
```

45.29.2.3 Qore::HTTPClient::clearStats ( )

Clears performance statistics.

Example:

```
$httpclient.clearStats();
```

Since

Qore 0.8.9

See also

HTTPClient::getUsageInfo()

45.29.2.4 nothing Qore::HTTPClient::clearUserPassword ( )

Clears the username and password for the connection.

Call this method after calling HTTPClient::setURL() to clear any authentication information present in the URL used in HTTPClient::setURL()

Example:

```
$httpclient.clearUserPassword();
```

45.29.2.5 nothing Qore::HTTPClient::clearWarningQueue ( )

Removes any warning Queue object from the Socket.

Example:

```
$httpclient.clearWarningQueue();
```

See also

HTTPClient::setWarningQueue()

Since

Qore 0.8.9

45.29.2.6 nothing Qore::HTTPClient::connect ( )

Connects to the remote socket; SSL/TLS negotiation is performed if required.

If the protocol indicates that a secure connection should be established (or HTTPClient::setSecure() was called previsouly), SSL/TLS negotiation will be attempted.

If the `TCP_NODELAY` flag has been set (see HTTPClient::setNoDelay()), then after a successful connection to the remote socket, this option will be set on the socket. If an error occurs setting the `TCP_NODELAY` option, the internal flag is set to false (use HTTPClient::getNoDelay() to check the flag's state) and the error code can be retrieved with errno().

Example:

```
$httpclient.connect();
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_R↩ESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED

Note

For possible exceptions, see the Socket::connect() method (or Socket::connectSSL() for secure connections).

45.29.2.7 Qore::HTTPClient::constructor (hash *opts*)

Creates the HTTPClient object based on the option parameter passed.

To connect, call any method that requires a connection and an implicit connection is established, or call HTTPClient::connect().

Example:

```
my HTTPClient $httpclient(("url":"http://hostname:8080/path"));
```

Parameters

opts	sets options and changes default behaviour for the object, etc; key names are case-sensitive and therefore must all be in lower-case:
	• `url`: A string giving the URL to connect to
	• `default_port`: The default port number to connect to if none is given in the URL
	• `protocols`: A hash describing new protocols, the key is the protocol name and the value is either an integer giving the default port number or a hash with `"port"` and `"ssl"` keys giving the default port number and a boolean value to indicate that an SSL connection should be established
	• `http_version`: Either `"1.0"` or `"1.1"` for the claimed HTTP protocol version compliancy in outgoing message headers
	• `default_path`: The default path to use for new connections if a path is not otherwise specified in the connection URL
	• `max_redirects`: The maximum number of redirects before throwing an exception (the default is 5)
	• `proxy`: The proxy URL for connecting through a proxy
	• `timeout`: The timeout value in milliseconds (also can be a relative date-time value for clarity, ex: `5m`)
	• `connect_timeout`: The timeout value in milliseconds for establishing a new socket connection (also can be a relative date-time value for clarity, ex: `30s`)
	• `additional_methods`: Optional hash with more but not-HTTP-standardized methods to handle. It allows to create various HTTP extensions like e.g. WebDAV. The hash uses method name as a key and value is a boolean `True` or `False` indicating if the HTTPClient should post the message body as well. Example:
	`# add new HTTP methods for WebDAV. Both of them require body posting to the server`
	`("additional_methods" : ( "PROPFIND" : True, "MKCOL": True ) );`

Exceptions

HTTP-CLIENT-OPTION-ERROR	invalid or unknown option passed in option hash
HTTP-CLIENT-URL-ERROR	invalid URL string

HTTP-CLIENT-UNKNOWN-PROTOCOL	unknown protocol passed in URL

Note

URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme:///socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: "`http://socket=%2ftmp%socket-dir%2fsocket-file-1/` this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

45.29.2.8 Qore::HTTPClient::constructor ( )

Creates the HTTPClient object.

Example:

```
my HTTPClient $httpclient();
```

45.29.2.9 Qore::HTTPClient::copy ( )

Copying objects of this class is not supported, an exception will be thrown.

Exceptions

HTTPCLIENT-COPY-ERROR	copying HTTPClient objects is not yet supported

45.29.2.10 Qore::HTTPClient::destructor ( )

Destroys the HTTPClient object and closes any open connections.

Example:

```
delete $httpclient;
```

45.29.2.11 nothing Qore::HTTPClient::disconnect ( )

Disconnects from the remote socket if a connection is established (otherwise does nothing)

Example:

```
$httpclient.disconnect();
```

45.29.2.12 __7_ string Qore::HTTPClient::get (string *path*, __7_ hash *headers*, __7_ reference *info*)

Sends an HTTP `GET` request and returns the message body received as a string or NOTHING if no message body is received.

In order to get the headers and the body, use the HTTPClient::send() method instead.

If no connection has already been established, an internal call to HTTPClient::connect() will be made before sending the request.

If any content encoding is used for the message body in the reply, the content is decoded and returned as a string; if the content encoding uses an unknown method, then an exception is thrown.

Example:

```
my *string $html = $httpclient.get("/path/file.html");
```

Parameters

path	the path for the message (i.e. "/path/resource?method&param=value")
headers	an optional hash of headers to include in the message
info	an optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request

Returns

the message body in the reply to this message or NOTHING in case of an error or an erroneous reply by the server with no body

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED, EVENT_HTTP_SEND_MESSAGE, EVENT_PACKET_SENT, EVENT_HTTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_HTTP_CONTENT_LENGTH, EVENT_HTTP_CHUNKED_START, EVENT_HTTP_CHUNKED_END, EVENT_HTTP_CHUNKED_DATA_RECEIVED, EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED, EVENT_HTTP_REDIRECT

Exceptions

HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the return value of the send() method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.13 __7_ string Qore::HTTPClient::getConnectionPath ( )

Returns the current connection path set in the URL.

Returns

the current connection path set in the URL

Code Flags:

CONSTANT

Example:

```
my *string $path = $httpclient.getConnectionPath();
```

45.29.2.14 int Qore::HTTPClient::getConnectTimeout ( )

Returns the connect timeout as an integer in milliseconds.

Returns

Returns the connect timeout as an integer in milliseconds; negative numbers mean the system default timeout is used

Code Flags:

CONSTANT

Example:

```
my int $to = $httpclient.getConnectTimeout();
```

45.29.2.15 string Qore::HTTPClient::getDefaultPath ( )

Returns the default path used by the object if no path is set in the URL.

Returns

the default path used by the object if no path is set in the URL

Code Flags:

CONSTANT

Example:

```
my *string $def_path = $httpclient.getDefaultPath();
```

45.29.2.16 string Qore::HTTPClient::getEncoding ( )

Returns the character encoding used for the object.

Returns

the character encoding used for the object

Code Flags:

CONSTANT

Example:

```
my string $encoding = $httpclient.getEncoding();
```

45.29.2.17 string Qore::HTTPClient::getHTTPVersion ( )

Returns the HTTP protocol version string used in outgoing messages.

Returns

the HTTP protocol version string used in outgoing messages

Code Flags:

CONSTANT

Example:

```
my string $version = $httpclient.getHTTPVersion();
```

45.29.2.18 int Qore::HTTPClient::getMaxRedirects ( )

Returns the current `max_redirects` value for the object (the maximum number of HTTP redirects that will be processed before an exception is raised)

Returns

the current `max_redirects` value for the object (the maximum number of HTTP redirects that will be processed before an exception is raised)

Code Flags:

CONSTANT

Example:

```
my int $mr = $httpclient.getMaxRedirects();
```

45.29.2.19 bool Qore::HTTPClient::getNoDelay ( )

Returns the `TCP_NODELAY` setting for the HTTPClient object.

Code Flags:

CONSTANT

Example:

```
my bool $b = $httpclient.getNoDelay();
```

See also

also HTTPClient::setNoDelay()

45.29.2.20 __7_ string Qore::HTTPClient::getProxyURL ( )

Returns the current proxy URL as a string or NOTHING if no proxy URL is set.

Returns

the current proxy URL as a string or NOTHING if no proxy URL is set

Code Flags:

CONSTANT

Example:

```
my *string $proxy_url = $httpclient.getProxyURL();
```

45.29.2.21 int Qore::HTTPClient::getTimeout ( )

Returns the default I/O timeout as an integer in milliseconds.

Returns

the default I/O timeout as an integer in milliseconds; 0 means immediate timeout (when reading only returns data if it is already available), and negative numbers mean never timeout

Code Flags:

CONSTANT

Example:

```
my int $timeout = $httpclient.getTimeout();
```

45.29.2.22 __7_ string Qore::HTTPClient::getURL ( )

Returns the current URL.

Returns

the current URL

Code Flags:

CONSTANT

Example:

```
my *string $url = $httpclient.getURL();
```

45.29.2.23 hash Qore::HTTPClient::getUsageInfo ( )

Returns performance statistics for the socket.

Code Flags:

CONSTANT

Example:

```
my hash $h = $httpclient.getUsageInfo();
```

Returns

a hash with the following keys:

- `"bytes_sent"`: an integer giving the total amount of bytes sent
- `"bytes_recv"`: an integer giving the total amount of bytes received
- `"us_sent"`: an integer giving the total number of microseconds spent sending data
- `"us_recv"`: an integer giving the total number of microseconds spent receiving data
- `"arg"`: (only if warning values have been set with HTTPClient::setWarningQueue()) the optional argument for warning hashes
- `"timeout"`: (only if warning values have been set with HTTPClient::setWarningQueue()) the warning timeout in microseconds
- `"min_throughput"`: (only if warning values have been set with HTTPClient::setWarningQueue()) the minimum warning throughput in bytes/sec

Since

Qore 0.8.9

See also

HTTPClient::clearStats()

45.29.2.24 hash Qore::HTTPClient::head (string *path*, __7_ hash *headers*, __7_ reference *info*)

Sends an HTTP `HEAD` request and returns as hash of the headers received.

If no connection is established, an internal call to HTTPClient::connect() will be made before sending the message.

Example:

```
my hash $msg = $httpclient.head("/path");
```

Parameters

path	the path for the message (i.e. `"/path/resource?method&param=value"`)
headers	an optional hash of headers to include in the message
info	an optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request

Returns

the headers received from the HTTP server with all key names converted to lower-case

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED, EVENT_HTTP_SEND_MESSAGE, EVENT_PACKET_SENT, EVENT_HTTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_HTTP_REDIRECT

Exceptions

HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the return value of the send() method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.25 bool Qore::HTTPClient::isConnected ( )

Returns True or False giving the current connection state.

Returns

True or False giving the current connection state

Code Flags:

CONSTANT

Example:

```
my bool $b = $httpclient.isConnected();
```

45.29.2.26 bool Qore::HTTPClient::isProxySecure ( )

Returns the SSL/TLS flag for the next proxy connection.

Returns

the SSL/TLS flag for the next proxy connection

Code Flags:

CONSTANT

Example:

```
my bool $b = $httpclient.isProxySecure();
```

45.29.2.27 bool Qore::HTTPClient::isSecure ( )

Returns True if the current connection is encrypted, False if not.

Returns

True if the current connection is encrypted, False if not

Code Flags:

CONSTANT

Example:

```
if ($httpclient.isSecure())
    printf("secure connection: %s %s\n", $httpclient.getSSLCipherName(), $httpclient.
      getSSLCipherVersion());
```

45.29.2.28 __7_ string Qore::HTTPClient::post (string *path*, string *body*, __7_ hash *headers*, __7_ reference *info*)

Sends an HTTP `POST` request with a message body and returns the message body received as a string or NOTHING if no message body is received.

In order to get the headers and the body in the response, use the HTTPClient::send() method instead.

If no connection is established, an internal call to HTTPClient::connect() will be made before sending the message.

Example:

```
$httpclient.post("/path", $body);
```

Parameters

path	the path for the message (i.e. "`/path/resource?method&param=value`")
body	the string to use as the message body
headers	an optional hash of headers to include in the message
info	an optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request

Returns

the message body in the reply to this message or NOTHING in case no body was present in the response

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED, EVENT_HTTP_SEND_MESSAGE, EVENT_PACKET_SENT, EVENT_HTTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_HTTP_CONTENT_LENGTH, EVENT_HTTP_CHUNKED_START, EVENT_HTTP_CHUNKED_END, EVENT_HTTP_CHUNKED_DATA_RECEIVED, EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED, EVENT_HTTP_REDIRECT

Exceptions

HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the `"arg"` key of the exception hash will be set to a hash equal to the return value of the send() method including a `"status_code"` key (giving the status code) and a `"body"` key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.29 __7_ string Qore::HTTPClient::post (string *path*, __7_ binary *body*, __7_ hash *headers*, __7_ reference *info*)

Sends an HTTP `POST` request with a message body and returns the message body received as a string or NOTHING if no message body is received.

In order to get the headers and the body in the response, use the HTTPClient::send() method instead.

If no connection is established, an internal call to HTTPClient::connect() will be made before sending the message.

Example:

```
$httpclient.post("/path", $body);
```

Parameters

path	the path for the message (i.e. `"/path/resource?method&param=value"`)
body	the optional data to use as the message body
headers	an optional hash of headers to include in the message
info	an optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request

Returns

the message body in the reply to this message or NOTHING in case no body was present in the response

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED, EVENT_HTTP_SEND_MESSAGE, EVENT_PACKET_SENT, EVENT_HTTP_MESSAGE_RECEIVED, EVENT_PACKET_READ, EVENT_HTTP_CONTENT_LENGTH, EVENT_HTTP_CHUNKED_START, EVENT_HTTP_CHUNKED_END, EVENT_HTTP_CHUNKED_DATA_RECEIVED, EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED, EVENT_HTTP_REDIRECT

Exceptions

HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "`arg`" key of the exception hash will be set to a hash equal to the return value of the send() method including a "`status_code`" key (giving the status code) and a "`body`" key (giving the message body returned by the server)
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.30 hash Qore::HTTPClient::send (string *body*, string *method*, __7_ string *path*, __7_ hash *headers*, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and optional message body and returns headers and any body received as a response in a hash format.

If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.send($body, "POST", "/path", ("Content-Type":"application/x-yaml"));
```

Parameters

body	The message body to send
method	The name of the HTTP method ("GET", "POST", "HEAD", "OPTIONS", "PUT", "DELETE", "TRACE", or "CONNECT"). Additional methods can be added in the constructor as a additional_methods option.
path	The path for the message (i.e. "/path/resource?method&param=value")
headers	An optional hash of headers to include in the message.
getbody	If this argument is True, then the object will try to receive a message body even if no "Content-Length" header is present in the response. Use this only with broken servers that send message bodies without a "Content-Length" header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Returns

The headers received from the HTTP server with all key names converted to lower-case. The message body (if any) will be assigned to the value of the "body" key and the HTTP status will be assigned to the "status_code" key.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code $<$ 100 or $>$ 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the normal return value of this method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.31 hash Qore::HTTPClient::send (__7_ binary *body*, string *method*, __7_ string *path*, __7_ hash *headers*, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and optional message body and returns headers and any body received as a response in a hash format.

If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.send($body, "POST", "/path", ("Content-Type":"application/x-yaml"));
```

Parameters

body	The message body to send; pass NOTHING (no value) to send no body
method	The name of the HTTP method ("GET", "POST", "HEAD", "OPTIONS", "PUT", "DELETE", "TRACE", or "CONNECT"). Additional methods can be added in the constructor as a additional_methods option.
path	The path for the message (i.e. "/path/resource?method&param=value")
headers	An optional hash of headers to include in the message.
getbody	If this argument is True, then the object will try to receive a message body even if no "Content-Length" header is present in the response. Use this only with broken servers that send message bodies without a "Content-Length" header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Returns

The headers received from the HTTP server with all key names converted to lower-case. The message body (if any) will be assigned to the value of the "body" key and the HTTP status will be assigned to the "status_code" key.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the normal return value of this method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

45.29.2.32 nothing Qore::HTTPClient::sendWithCallbacks (code *scb*, code *rcb*, string *method*, __7_ string *path*, __7_ hash *headers*, timeout *timeout_ms* = 0, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and chunked message body as given by a send callback; headers and any body received are returned through a receive callback.

This method is useful for sending chunked message data where the response is also a sent with chunked transfer encoding; chunks are sent to the receive callback as soon as they are received. If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.sendWithCallbacks($send_callback, $rcv_callback, "POST", "/path", ("
      Content-Type":"application/x-yaml"));
```

Parameters

scb	The callback giving the chunked HTTP data to send; this callback must return either a string or a binary value each time it is called to give the chunked data to send; when all data has been sent, then a hash of message trailers can be sent or simply NOTHING which will close the chunked message
rcb	The receive callback for the data received; first this method is called with a hash of the message headers, and then with any message body; if a chunked HTTP message is received, then the callback is called once for each chunk; when the message has been received, then the receive callback is called with a hash representing any trailer data received in a chunked transfer or NOTHING if the data was received in a normal message body or if there was no trailer data in a chunked transfer. The argument to this callback is always a hash; data calls have the following keys: • `"data"`: the string or binary data • `"chunked"`: True if the data was received with chunked transfer encoding, False if not Header or trailer data is placed in a hash with the following keys: • `"hdr"`: this can be assigned to NOTHING for the trailer hash if the data was not sent chunked or no trailers were included in a chunked message • `"obj"`: this is the owning object (so socket parameters can be changed based on headers received, such as, for example, socket character encoding) • `"send_aborted"`: this is set to True if a response header was set while sending an outgoing chunked message

method	The name of the HTTP method ("GET", "POST", "HEAD", "OPTIONS", "PUT", "DELETE", "TRACE", or "CONNECT"). Additional methods can be added in the constructor as a additional_methods option.
path	The path for the message (i.e. "/path/resource?method&param=value")
headers	An optional hash of headers to include in the message.
timeout_ms	the timeout in milliseconds for the socket I/O operations; 0 means use the default timeout value
getbody	If this argument is True, then the object will try to receive a message body even if no "Content-Length" header is present in the response. Use this only with broken servers that send message bodies without a "Content-Length" header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the normal return value of this method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

- The "Transfer-Encoding: chunked" header is automatically set with this method if no "Transfer-Encoding" header is already present
- if a response is received while the chunked send operation is still in progress, an error is assumed, the send operation is aborted, and the response header is read immediately and the send_aborted flag is set in the argument to *rcb*
- For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

Since

Qore 0.8.10

45.29.2.33 nothing Qore::HTTPClient::sendWithRecvCallback (code *rcb*, string *body*, string *method*, __7_ string *path*, __7_ hash *headers*, timeout *timeout_ms* = 0, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and optional message body; headers and any body received are returned through a receive callback.

This method is useful for receiving chunked message data in real time; chunks are sent to the receive callback as soon as they are received. If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.sendWithRecvCallback($rcv_callback, $data, "POST", "/path", ("Content-Type":
      "application/x-yaml"));
```

Parameters

rcb	The receive callback for the data received; first this method is called with a hash of the message headers, and then with any message body; if a chunked HTTP message is received, then the callback is called once for each chunk; when the message has been received, then the receive callback is called with a hash representing any trailer data received in a chunked transfer or NOTHING if the data was received in a normal message body or if there was no trailer data in a chunked transfer. The argument to this callback is always a hash; data calls have the following keys: • `"data"`: the string or binary data • `"chunked"`: True if the data was received with chunked transfer encoding, False if not Header or trailer data is placed in a hash with the following keys: • `"hdr"`: this can be assigned to NOTHING for the trailer hash if the data was not sent chunked or no trailers were included in a chunked message • `"obj"`: this is the owning object (so socket parameters can be changed based on headers received, such as, for example, socket character encoding)
body	The message body to send
method	The name of the HTTP method (`"GET"`, `"POST"`, `"HEAD"`, `"OPTIONS"`, `"PUT"`, `"DELETE"`, `"TRACE"`, or `"CONNECT"`). Additional methods can be added in the constructor as a `additional_methods` option.
path	The path for the message (i.e. `"/path/resource?method&param=value"`)
headers	An optional hash of headers to include in the message.
timeout_ms	the timeout in milliseconds for the socket I/O operations; 0 means use the default timeout value
getbody	If this argument is True, then the object will try to receive a message body even if no `"Content-Length"` header is present in the response. Use this only with broken servers that send message bodies without a `"Content-Length"` header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the `"arg"` key of the exception hash will be set to a hash equal to the normal return value of this method including a `"status_code"` key (giving the status code) and a `"body"` key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

Since

Qore 0.8.10

45.29.2.34 nothing Qore::HTTPClient::sendWithRecvCallback (code *rcb*, __7_ binary *body*, string *method*, __7_ string *path*, __7_ hash *headers*, timeout *timeout_ms* = 0, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and optional message body; headers and any body received are returned through a receive callback.

This method is useful for receiving chunked message data in real time; chunks are sent to the receive callback as soon as they are received. If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.sendWithRecvCallback($rcv_callback, $data, "POST", "/path", ("Content-Type":
      "application/x-yaml"));
```

Parameters

rcb	The receive callback for the data received; first this method is called with a hash of the message headers, and then with any message body; if a chunked HTTP message is received, then the callback is called once for each chunk; when the message has been received, then the receive callback is called with a hash representing any trailer data received in a chunked transfer or NOTHING if the data was received in a normal message body or if there was no trailer data in a chunked transfer. The argument to this callback is always a hash; data calls have the following keys: • `"data"`: the string or binary data • `"chunked"`: True if the data was received with chunked transfer encoding, False if not Header or trailer data is placed in a hash with the following keys: • `"hdr"`: this can be assigned to NOTHING for the trailer hash if the data was not sent chunked or no trailers were included in a chunked message • `"obj"`: this is the owning object (so socket parameters can be changed based on headers received, such as, for example, socket character encoding)
body	The message body to send; pass NOTHING (no value) to send no body
method	The name of the HTTP method (`"GET"`, `"POST"`, `"HEAD"`, `"OPTIONS"`, `"PUT"`, `"DELETE"`, `"TRACE"`, or `"CONNECT"`). Additional methods can be added in the constructor as a `additional_methods` option.
path	The path for the message (i.e. `"/path/resource?method&param=value"`)
headers	An optional hash of headers to include in the message.
timeout_ms	the timeout in milliseconds for the socket I/O operations; 0 means use the default timeout value
getbody	If this argument is True, then the object will try to receive a message body even if no `"Content-Length"` header is present in the response. Use this only with broken servers that send message bodies without a `"Content-Length"` header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the `"arg"` key of the exception hash will be set to a hash equal to the normal return value of this method including a `"status_code"` key (giving the status code) and a `"body"` key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

Since

Qore 0.8.10

45.29.2.35 hash Qore::HTTPClient::sendWithSendCallback (code *scb*, string *method*, __7_ string *path*, __7_ hash *headers*, timeout *timeout_ms* = 0, softbool *getbody* = False, __7_ reference *info*)

Sends an HTTP request with the specified method and chunked message body as given by a send callback and returns headers and any body received as a response in a hash format.

If a connection has not already been established, an internal call to HTTPClient::connect() will be made before sending the message

Example:

```
my hash $msg = $httpclient.sendWithSendCallback($callback, "POST", "/path", ("Content-Type":"
      application/x-yaml"));
```

Parameters

scb	The callback giving the chunked HTTP data to send; this callback must return either a string or a binary value each time it is called to give the chunked data to send; when all data has been sent, then a hash of message trailers can be sent or simply NOTHING which will close the chunked message
method	The name of the HTTP method ("GET", "POST", "HEAD", "OPTIONS", "PUT", "DELETE", "TRACE", or "CONNECT"). Additional methods can be added in the constructor as a additional_methods option.
path	The path for the message (i.e. "/path/resource?method&param=value")
headers	An optional hash of headers to include in the message.
timeout_ms	the timeout in milliseconds for the socket I/O operations; 0 means use the default timeout value
getbody	If this argument is True, then the object will try to receive a message body even if no "Content-Length" header is present in the response. Use this only with broken servers that send message bodies without a "Content-Length" header.
info	An optional reference to an lvalue that will be used as an output variable giving a hash of request headers and other information about the HTTP request.

Returns

The headers received from the HTTP server with all key names converted to lower-case. The message body (if any) will be assigned to the value of the "body" key and the HTTP status will be assigned to the "status_code" key.

Exceptions

HTTP-CLIENT-METHOD-ERROR	invalid/unknown HTTP method passed
HTTP-CLIENT-REDIRECT-ERROR	invalid redirect location given by remote
HTTP-CLIENT-MAXIMUM-REDIRECTS-EXCEEDED	maximum redirect count exceeded
HTTP-CLIENT-RECEIVE-ERROR	unknown content encoding received or status error communicating with HTTP server (status code < 100 or > 299); in case of a status error the "arg" key of the exception hash will be set to a hash equal to the normal return value of this method including a "status_code" key (giving the status code) and a "body" key (giving the message body returned by the server)
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	There was an error sending the data
SOCKET-CLOSED	The remote end closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	Data transmission or reception for a single send() or recv() action exceeded the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	Invalid HTTP data was received

Note

- The "Transfer-Encoding: chunked" header is automatically set with this method if no "Transfer-Encoding" header is already present
- For possible exceptions when implicitly establishing a connection, see the Socket::connect() method (or Socket::connectSSL() for secure connections)

Since

Qore 0.8.10

45.29.2.36 nothing Qore::HTTPClient::setConnectTimeout (timeout *timeout_ms* = -1)

Sets the connect timeout in milliseconds.

Parameters

timeout_ms	the connect timeout in milliseconds; negative numbers mean use the default system connect timeout. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 30s = 30 seconds, etc.).

Example:

```
$httpclient.setConnectTimeout(2m);
```

45.29.2.37 nothing Qore::HTTPClient::setDefaultPath (__7_ string *path*)

Sets the default path used by the object if no path is set in the URL.

Parameters

path	the default path value to set or if NOTHING then clears the path

Example:

```
$httpclient.setDefaultPath();
```

45.29.2.38 nothing Qore::HTTPClient::setEncoding (string *encoding*)

Sets the string encoding for the object; any strings deserialized with this object will be tagged with this character encoding.

Parameters

encoding	the string encoding for the object; any strings deserialized with this object will be tagged with this character encoding

Example:

```
$httpclient.setEncoding("UTF-8");
```

45.29.2.39 nothing Qore::HTTPClient::setEventQueue ( )

Clears any Queue object that may be set on the HTTPClient object so that I/O events are no longer captured on the object.

Example:

```
$httpclient.setEventQueue();
```

45.29.2.40 nothing Qore::HTTPClient::setEventQueue (Qore::Thread::Queue *queue*)

Sets a Queue object to receive HTTPClient and Socket events.

Parameters

queue	a Queue object to receive HTTPClient and Socket events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Example:

```
$httpclient.setEventQueue($queue);
```

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

See also

event_handling for more information

45.29.2.41 nothing Qore::HTTPClient::setHTTPVersion (string *ver*)

Sets the HTTP protocol version string for headers in outgoing messages, allowed values are "1.0" and "1.1".

Parameters

ver	"1.0" or "1.1" for the HTTP protocol compliance version

Example:

```
$httpclient.setHTTPVersion("1.1");
```

Exceptions

HTTP-VERSION-ERROR	invalid HTTP version passed (allowed values: "1.0" and "1.1")

45.29.2.42 nothing Qore::HTTPClient::setMaxRedirects (softint *mr* = 0)

Updates the setting for the max_redirects value for the object (maximum number of HTTP redirects that will be processed before an exception is raised)

Parameters

mr	the setting for the maximum number of HTTP redirects that will be processed before an exception is raised

Example:

```
$httpclient.setMaxRedirects(5);
```

See also

HTTPClient::getMaxRedirects() to retrieve this value

45.29.2.43 int Qore::HTTPClient::setNoDelay (softbool *b* = True)

Sets the TCP_NODELAY setting for the object.

When this setting is True, then data will be immediately sent out over the HTTPClient object's socket, when it is False, then data transmission may be delayed to be packaged with other data for the same target.

Delayed data transmissions may cause problems when the sender immediately closes the socket after sending data; in this case the receiver may not get the data even though the send succeeded.

Note that if no value is given to the method, the argument will be assumed to be True, and output buffering will be turned off for the HTTPClient object.

If the socket is not connected when this call is made, then an internal flag is set and the TCP_NODELAY option is enabled when the next connection is established. If the socket is connected, then if an error occurs setting the TCP_NODELAY option on the socket, this method will return a non-zero error code; the actual error can be checked with the errno() function.

Example:

```
$httpclient.setNoDelay(True);
```

Parameters

b	the TCP_NODELAY setting for the object

See also

HTTPClient::getNoDelay()

45.29.2.44 Qore::HTTPClient::setPersistent ( )

temporarily disables implicit reconnections; must be called when the server is already connected

Example:

```
$httpclient.connect();
$httpclient.setPersistent();
```

The persistent flag is automatically reset to False whenever the connection is closed; it must be called manually for every connection to turn off implicit reconnections.

To turn off the persistent flag manually, call HTTPClient::disconnect()

Since

Qore 0.8.10

45.29.2.45 nothing Qore::HTTPClient::setProxySecure (softbool *b* = True)

Sets the SSL/TLS flag for the next connection to the proxy.

Example:

```
$httpclient.setProxySecure(True);
```

See also

HTTPClient::isProxySecure() to check the flag

45.29.2.46 nothing Qore::HTTPClient::setProxyURL ( )

Clears the new proxy URL value for the next connection.

This variant of the method is equivalent to HTTPClient::clearProxyURL()

Example:

```
$httpclient.setProxyURL();
```

45.29.2.47 nothing Qore::HTTPClient::setProxyURL (string *url*)

Sets a new proxy URL value for the next connection.

Parameters

url	the new proxy URL value for the next connection

Example:

```
$httpclient.setProxyURL("http://user:password@proxy_host:8080/path");
```

Exceptions

HTTP-CLIENT-URL-ERR↩ OR	invalid proxy URL string; invalid authorization credentials in proxy URL (username without password or vice-versa)
HTTP-CLIENT-PROXY-P↩ ROTOCOL-ERROR	unknown protocol passed in URL

Note

URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme↩ ://socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: "`http://socket=%2ftmp%socket-dir%2fsocket-file-1/` this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

45.29.2.48 nothing Qore::HTTPClient::setProxyUserPassword (**string** *user,* **string** *pass*)

Sets the username and password for the connection to the proxy; call after HTTPClient::setProxyURL()

Call this method after calling HTTPClient::setProxyURL() to set proxy authentication information when not present in the URL used in HTTPClient::setProxyURL()

Example:

```
$httpclient.setProxyUserPassword($user, $pass);
```

Parameters

user	the username to use for proxy authentication in the next HTTP connection
pass	the password to use for proxy authentication in the next HTTP connection

45.29.2.49 nothing Qore::HTTPClient::setProxyUserPassword ( )

Clears the username and password for the next proxy connection.

Call this method after calling HTTPClient::setProxyURL() to clear any proxy authentication information present in the URL used in HTTPClient::setProxyURL()

This variant of the method is equivalent to HTTPClient::clearProxyUserPassword()

Example:

```
$httpclient.setProxyUserPassword();
```

See also

HTTPClient::clearProxyUserPassword()

45.29.2.50 nothing Qore::HTTPClient::setSecure (softbool *secure* = **True**)

Sets the object to make a secure SSL/TLS connection on the next connect if the passed argument is True, or an unencrypted cleartext connection if it is False.

This method overrides the default behaviour for the protocol set for the object

Note that the behavior of this method when called with no argument changed in version 0.8.0; prior to version 0.8.0 calling this method with no argument would turn off secure mode; the behavior was changed to the current functionality in order to make the usage of this method consistent with other methods of the same name and to make it more logical.

Example:

```
$httpclient.setSecure(True);
```

Parameters

secure	if True, a SSL/TLS connection will be attempted on the next connection. If False, an unencrypted cleartext connection will be established

45.29.2.51 nothing Qore::HTTPClient::setTimeout (timeout *timeout_ms* = 0)

Sets the default I/O timeout value in milliseconds.

Parameters

timeout_ms	0 means immediate timeout (when reading will return data only if it is already available), and negative numbers mean never timeout. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.).

Example:

```
$httpclient.setTimeout(2m);
```

45.29.2.52 Qore::HTTPClient::setURL (string *url*)

Sets a new URL value for the next connection.

To retrieve the current URL value, use the HTTPClient::getURL() method

Example:

```
$httpclient.setURL("https://user:password@hostname:8080/path");
```

Parameters

url	the new URL for the object

Exceptions

HTTP-CLIENT-URL-ERROR	invalid URL string; invalid authorization credentials in URL (username without password or vice-versa)
HTTP-CLIENT-UNKNOWN-PROTOCOL	unknown protocol (scheme) passed in URL

See also

HTTPClient::getURL()

Note

URLs with UNIX sockets are generally supported in Qore with the following syntax: `scheme://socket=<url_encoded_path>/path`, where `url_encoded_path` is a path with URL-encoding as performed by encode_url(); for example: `"http://socket=%2ftmp%socket-dir%2fsocket-file-1/` this allows a filesystem path to be used in the host portion of the URL and for the URL to include a URL path as well.

45.29.2.53 nothing Qore::HTTPClient::setUserPassword (string *user*, string *pass*)

Sets the username and password for the connection; call after HTTPClient::setURL()

Call this method after calling HTTPClient::setURL() to set authentication information when not present in the URL used in HTTPClient::setURL()

Parameters

user	the username to use for authentication in the next HTTP connection
pass	the password to use for authentication in the next HTTP connection

Example:

```
$httpclient.setUserPassword($user, $pass);
```

45.29.2.54 nothing Qore::HTTPClient::setUserPassword ( )

Clears the username and password for the connection.

Call this method after calling HTTPClient::setURL() to clear any authentication information present in the URL used in HTTPClient::setURL()

This variant of the method is equivalent to HTTPClient::clearUserPassword()

Example:

```
$httpclient.setUserPassword();
```

See also

HTTPClient::clearUserPassword()

45.29.2.55 nothing Qore::HTTPClient::setWarningQueue (int *warning_ms*, int *warning_bs*, Queue *queue*, any *arg*, timeout *min_ms* = `1s`)

Sets a Queue object to receive socket warnings.

Example:

```
$httpclient.setWarningQueue(5000, 5000, $queue, "socket-1");
```

Parameters

warning_ms	the threshold in milliseconds for individual socket actions (send, receive, connect), if exceeded, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-OPERATION-WARNING"` • `"operation"`: a string giving the operation that caused the warning (ex: `"connect"`) • `"us"`: an integer giving the number of microseconds for the operation • `"timeout"`: an integer giving the warning threshold in microseconds • `"arg"`: if any `"arg"` argument is passed to the HTTPClient::setWarningQueue() method, it will be included in the warning hash here
warning_bs	value in bytes per second; if any call has performance below this threshold, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-THROUGHPUT-WARNING"` • `"dir"`: either `"send"` or `"recv"` depending on the direction of the data flow • `"bytes"`: the amount of bytes sent • `"us"`: an integer giving the number of microseconds for the operation • `"bytes_sec"`: a float giving the transfer speed in bytes per second • `"threshold"`: an integer giving the warning threshold in bytes per second • `"arg"`: if any `"arg"` argument is passed to the HTTPClient::setWarningQueue() method, it will be included in the warning hash here
queue	the Queue object to receive warning events
arg	an optional argument to be placed in the `"arg"` key in each warning hash (could be used to identify the socket for example)
min_ms	the minimum transfer time with a resolution of milliseconds for a transfer to be eligible for triggering a warning; transfers that take less than this period of time are not eligible for raising a warning

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set
SOCKET-SETWARNINGQUEUE-ERROR	at least one of *warning_ms* and *warning_bs* must be > 0

See also

HTTPClient::clearWarningQueue()

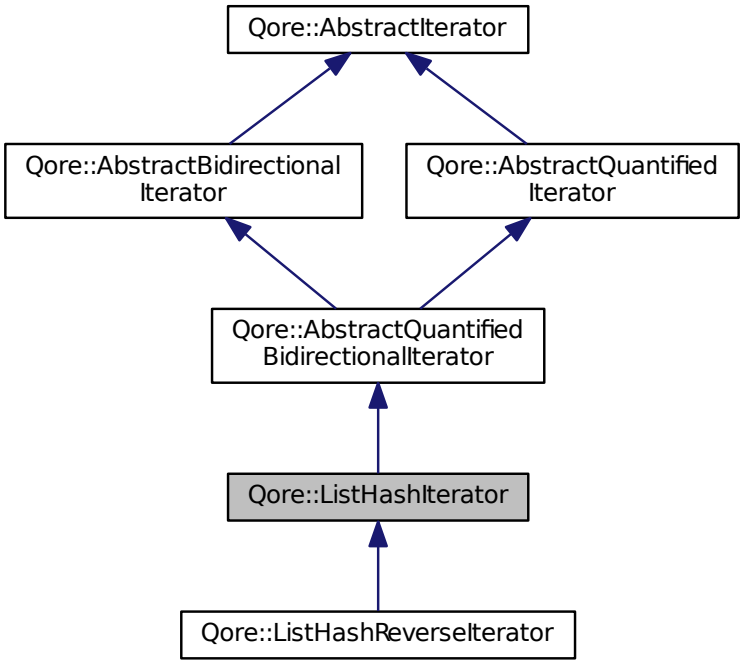
Since

Qore 0.8.9

45.30 Qore::ListHashIterator Class Reference

This class an iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return lists of columns where each list entry is a hash of the current column values.

Inheritance diagram for Qore::ListHashIterator:



Public Member Functions

- constructor (softlist l)

  Creates the hash list iterator object.
- copy ()

  Creates a copy of the ListHashIterator object, iterating the same object as the original and in the same position.
- bool empty ()

  returns True if the result list is empty; False if not
- bool first ()

  returns True if on the first element of the list
- any getKeyValue (string key)

  Returns the current value for the column given as an argument.
- hash getRow ()

  returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- hash getValue ()

  returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- int index ()

  returns the current iterator position in the list or -1 if not pointing at a valid element
- bool last ()

  returns True if on the last element of the list
- int max ()

  returns the number of elements in the list
- any memberGate (string key)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

- bool next ()

  Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

- bool prev ()

  Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

- reset ()

  Reset the iterator instance to its initial state.

- bool set (int pos)

  sets the new position in the result list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True

- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.30.1 Detailed Description

This class an iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return lists of columns where each list entry is a hash of the current column values.

Call ListHashIterator::next() to iterate through the lists of column values; do not use the iterator if ListHashIterator::next() returns False. A result list can be iterated in reverse order by calling ListHashIterator::prev() instead of ListHashIterator::next()

Example: ListHashIterator basic usage

```
my list $data = (
         ( "column1" : 1, "column2" : "a"),
         ( "column1" : 2, "column2" : "b"),
    );

my ListHashIterator $it($data);
while ($it.next()) {
    printf("iter %d: %n\n", $it.index(), $it.getValue());
}

iter 0: hash: (column1 : 1, column2 : "a")
iter 1: hash: (column1 : 2, column2 : "b")
```

Note

- In general, the ListHashIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

ListHashReverseIterator

45.30.2 Member Function Documentation

45.30.2.1 Qore::ListHashIterator::constructor (softlist /)

Creates the hash list iterator object.

Parameters

l	the list of hashes to iterate

Example:

```
my ListHashIterator $i($l);
```

Note

the constructor's argument is softlist so that it can also accept NOTHING

45.30.2.2 Qore::ListHashIterator::copy ( )

Creates a copy of the ListHashIterator object, iterating the same object as the original and in the same position.

Example:

```
my ListHashIterator $ni = $i.copy();
```

45.30.2.3 bool Qore::ListHashIterator::empty ( ) [virtual]

returns True if the result list is empty; False if not

Returns

True if the result list is empty; False if not

Code Flags:

CONSTANT

Example:

```
if ($i.empty())
    printf("the result list is empty\n");
```

Implements Qore::AbstractQuantifiedIterator.

45.30.2.4 bool Qore::ListHashIterator::first ( ) [virtual]

returns True if on the first element of the list

Returns

True if on the first element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ListHashReverseIterator.

45.30.2.5 any Qore::ListHashIterator::getKeyValue (string *key*)

Returns the current value for the column given as an argument.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.getKeyValue("value"));
}
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element or the list being iterated does not contain a hash element at the current iterator position or the given hash key does not exist

Note

ListHashIterator::memberGate() allows for the iterator to act as if it is a hash with members equal to the current row being iterated

45.30.2.6 hash Qore::ListHashIterator::getRow ( )

returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Note

equivalent to ListHashIterator::getValue()

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element or the list being iterated does not contain a hash element at the current iterator position

45.30.2.7 hash Qore::ListHashIterator::getValue ( ) [virtual]

returns the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current row value as a hash or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Note

- equivalent to ListHashIterator::getRow()

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element or the list being iterated does not contain a hash element at the current iterator position

Implements Qore::AbstractIterator.

45.30.2.8 int Qore::ListHashIterator::index ( )

returns the current iterator position in the list or -1 if not pointing at a valid element

Returns

the current iterator position in the list or -1 if not pointing at a valid element

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.30.2.9 bool Qore::ListHashIterator::last ( ) [virtual]

returns True if on the last element of the list

Returns

True if on the last element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ListHashReverseIterator.

45.30.2.10 int Qore::ListHashIterator::max ( )

returns the number of elements in the list

Returns

the number of elements in the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.30.2.11 any Qore::ListHashIterator::memberGate (string *key*)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.value);
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element or the list being iterated does not contain a hash element at the current iterator position or the given hash key does not exist

Note

equivalent to ListHashIterator::getKeyValue() when called explicitly

45.30.2.12 bool Qore::ListHashIterator::next ( ) [virtual]

Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

Reimplemented in Qore::ListHashReverseIterator.

45.30.2.13 bool Qore::ListHashIterator::prev ( ) [virtual]

Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractBidirectionalIterator.

Reimplemented in Qore::ListHashReverseIterator.

45.30.2.14 Qore::ListHashIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.30.2.15 bool Qore::ListHashIterator::set (int *pos*)

sets the new position in the result list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True

Parameters

pos	the new position for the iterator with 0 as the first element

Returns

False, meaning the iterator is not valid, otherwise it returns True

Example:

```
if (!$i.set($pos))
    throw "INVALID-POSITION", sprintf("%d is an invalid position", $pos);
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.30.2.16 bool Qore::ListHashIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

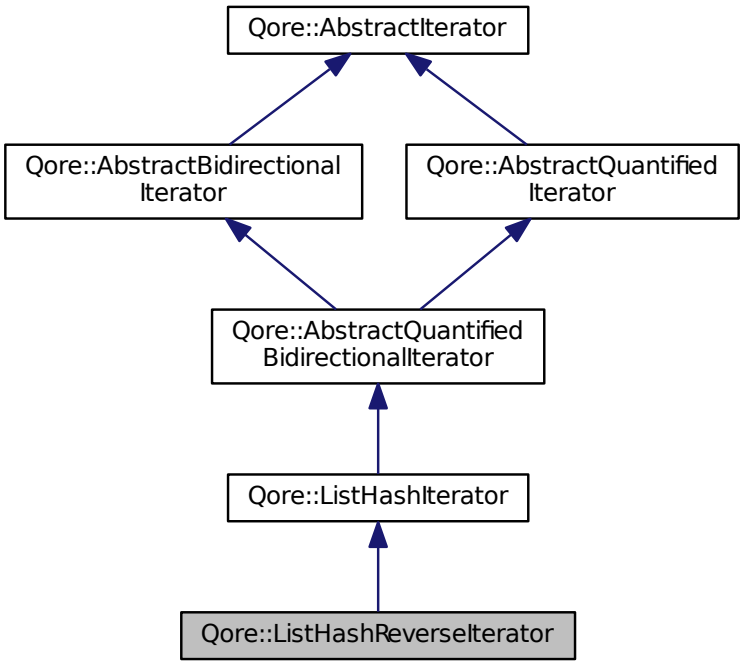
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.31 Qore::ListHashReverseIterator Class Reference

This class a reverse iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return hashes with keys giving column names where the key values are lists of column values.

Inheritance diagram for Qore::ListHashReverseIterator:



Public Member Functions

- constructor (softlist l)

  Creates the list hash iterator object.
- copy ()

  Creates a copy of the ListHashReverseIterator object, iterating the same object as the original and in the same position.
- bool first ()

  returns True if on the first element being iterated (ie the last element in the list)
- bool last ()

  returns True if on the last element being iterated (ie the first element in the list)
- any memberGate (string key)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

- bool next ()

  Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

- bool prev ()

  Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

45.31.1 Detailed Description

This class a reverse iterator class for lists of hashes as returned by Qore::SQL::Datasource::selectRows() and Qore::SQL::DatasourcePool::selectRows(), both of which return hashes with keys giving column names where the key values are lists of column values.

Call ListHashReverseIterator::next() to iterate through the lists of column values assigned to each hash key in reverse order; do not use the iterator if ListHashReverseIterator::next() returns False. A result list can be iterated in reverse order by calling ListHashReverseIterator::prev() instead of ListHashReverseIterator::next()

Example: ListHashReverseIterator basic usage

```
my list $data = (
         ( "column1" : 1, "column2" : "a"),
         ( "column1" : 2, "column2" : "b"),
    );

my ListHashReverseIterator $it($data);
while ($it.next()) {
    printf("iter %d: %n\n", $it.index(), $it.getValue());
}

iter 1: hash: (column1 : 2, column2 : "b")
iter 0: hash: (column1 : 1, column2 : "a")
```

Note

- In general, the ListHashReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- ListHashReverseIterator is functionally equivalent to ListHashIterator, but the effect of ListHashReverseIterator::next() and ListHashReverseIterator::prev() are the opposite of ListHashIterator::next() and ListHashIterator::prev(); that is ListHashReverseIterator::next() will iterate through the hash in reverse order, while ListHashReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for ListHashReverseIterator::first() and ListHashReverseIterator::last() are swapped in respect to ListHashIterator::first() and ListHashIterator::last().

See also

ListHashIterator

45.31.2 Member Function Documentation

45.31.2.1 Qore::ListHashReverseIterator::constructor (softlist *l*)

Creates the list hash iterator object.

Parameters

l	the list of hashes to iterate

Example:

```
my ListHashIterator $i($l);
```

Note

the constructor's argument is softlist so that it can also accept NOTHING

45.31.2.2 Qore::ListHashReverseIterator::copy ( )

Creates a copy of the ListHashReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my ListHashReverseIterator $ni = $i.copy();
```

45.31.2.3 bool Qore::ListHashReverseIterator::first ( ) [virtual]

returns True if on the first element being iterated (ie the last element in the list)

Returns

True if on the first element being iterated (ie the last element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Reimplemented from Qore::ListHashIterator.

45.31.2.4 bool Qore::ListHashReverseIterator::last ( ) [virtual]

returns True if on the last element being iterated (ie the first element in the list)

Returns

True if on the last element being iterated (ie the first element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Reimplemented from Qore::ListHashIterator.

45.31.2.5 any Qore::ListHashReverseIterator::memberGate (string *key*)

This method allows the iterator to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
while ($i.next()) {
    printf("%d: value: %y", $i.index(), $i.value);
}
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object
INVALID-ITERATOR	the iterator is not pointing at a valid element or the list being iterated does not contain a hash element at the current iterator position or the given hash key does not exist

Note

equivalent to ListHashIterator::getKeyValue() when called explicitly

45.31.2.6 bool Qore::ListHashReverseIterator::next ( ) [virtual]

Moves the current position to the next element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::ListHashIterator.

45.31.2.7 bool Qore::ListHashReverseIterator::prev ( ) [virtual]

Moves the current position to the previous element in the result list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the result list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + row %d: %y\n", $i.index(), $i.getValue());
}
```

Exceptions

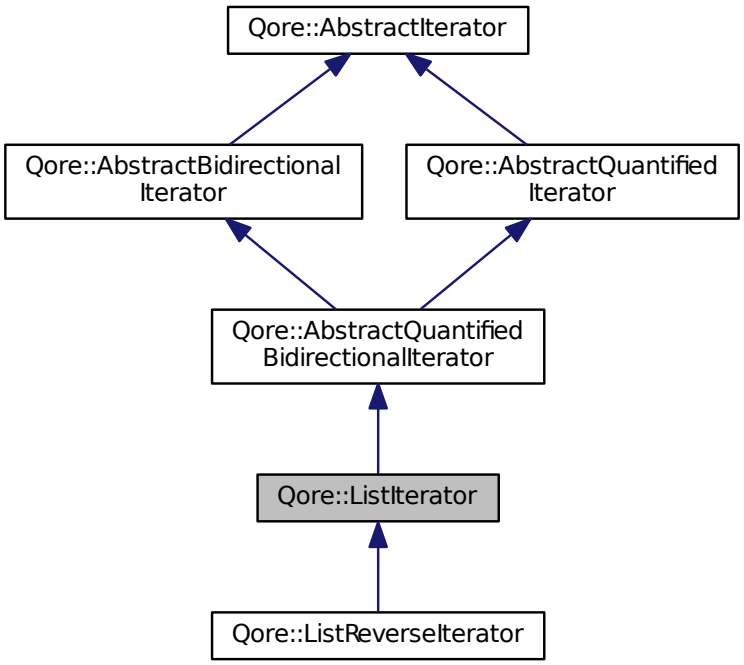
ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::ListHashIterator.

45.32 Qore::ListIterator Class Reference

This class an iterator class for lists.

Inheritance diagram for Qore::ListIterator:



Public Member Functions

- constructor (softlist l)

  Creates the list iterator object.
- copy ()

  Creates a copy of the ListIterator object, iterating the same object as the original and in the same position.
- bool empty ()

  returns True if the list is empty; False if not
- bool first ()

  returns True if on the first element of the list
- any getValue ()

  returns the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- int index ()

  returns the current iterator position in the list or -1 if not pointing at a valid element
- bool last ()

  returns True if on the last element of the list
- int max ()

  returns the number of elements in the list
- bool next ()

  Moves the current position to the next element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.
- bool prev ()

Moves the current position to the previous element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

- reset ()

  Reset the iterator instance to its initial state.

- bool set (int pos)

  sets the new position in the list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True

- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.32.1 Detailed Description

This class an iterator class for lists.

Call ListIterator::next() to iterate through the list; do not use the iterator if ListIterator::next() returns False. A list can be iterated in reverse order by calling ListIterator::prev() instead of ListIterator::next()

Example: ListIterator basic usage

```
my list $data = (1, "foo", 2);
my ListIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: 1
iter: "foo"
iter: 2
```

Note

- In general, the ListIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

ListReverseIterator

45.32.2 Member Function Documentation

45.32.2.1 Qore::ListIterator::constructor (softlist *l*)

Creates the list iterator object.

Parameters

l	the list to iterate

Example:

```
my ListIterator $li($l);
```

Note

the constructor's argument is softlist so that it can also accept NOTHING

45.32.2.2 Qore::ListIterator::copy ( )

Creates a copy of the ListIterator object, iterating the same object as the original and in the same position.

Example:

```
my ListIterator $ni = $i.copy();
```

45.32.2.3 bool Qore::ListIterator::empty ( ) [virtual]

returns True if the list is empty; False if not

Returns

True if the list is empty; False if not

Code Flags:

CONSTANT

Example:

```
if ($i.empty())
    printf("the list is empty\n");
```

Implements Qore::AbstractQuantifiedIterator.

45.32.2.4 bool Qore::ListIterator::first ( ) [virtual]

returns True if on the first element of the list

Returns

True if on the first element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ListReverseIterator.

45.32.2.5 any Qore::ListIterator::getValue ( ) [virtual]

returns the current value or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

45.32.2.6 int Qore::ListIterator::index ( )

returns the current iterator position in the list or -1 if not pointing at a valid element

Returns

the current iterator position in the list or -1 if not pointing at a valid element

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.32.2.7 bool Qore::ListIterator::last ( ) [virtual]

returns True if on the last element of the list

Returns

True if on the last element of the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Implements Qore::AbstractQuantifiedIterator.

Reimplemented in Qore::ListReverseIterator.

45.32.2.8 int Qore::ListIterator::max ( )

returns the number of elements in the list

Returns

the number of elements in the list

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    printf("+ %d/%d: %y\n", $i.index(), $i.max(), $i.getValue());
}
```

45.32.2.9 bool Qore::ListIterator::next ( ) [virtual]

Moves the current position to the next element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y\n", $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

Reimplemented in Qore::ListReverseIterator.

45.32.2.10 bool Qore::ListIterator::prev ( ) [virtual]

Moves the current position to the previous element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + %y\n", $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractBidirectionalIterator.

Reimplemented in Qore::ListReverseIterator.

45.32.2.11 Qore::ListIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.32.2.12 bool Qore::ListIterator::set (int *pos*)

sets the new position in the list; if the position is invalid then the method returns False, meaning the iterator is not valid, otherwise it returns True

Parameters

pos	the new position for the iterator with 0 as the first element

Returns

False, meaning the iterator is not valid, otherwise it returns True

Example:

```
if (!$i.set($pos))
    throw "INVALID-POSITION", sprintf("%d is an invalid position", $pos);
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.32.2.13 bool Qore::ListIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

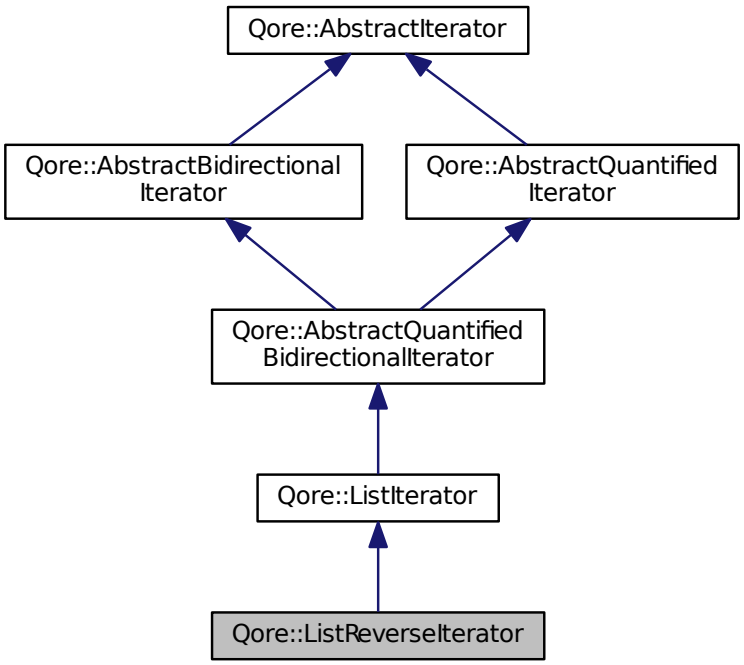
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.33 Qore::ListReverseIterator Class Reference

This class an iterator class for lists.

Inheritance diagram for Qore::ListReverseIterator:



Public Member Functions

- constructor (softlist l)

  Creates the list iterator object.
- copy ()

  Creates a copy of the ListReverseIterator object, iterating the same object as the original and in the same position.
- bool first ()

  returns True if on the first element iterated with this iterator (ie the last element in the list)
- bool last ()

  returns True if on the last element iterated with this iterator (ie the first element in the list)
- bool next ()

  Moves the current position to the previous element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

- bool prev ()

  Moves the current position to the next element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

45.33.1 Detailed Description

This class an iterator class for lists.

Call ListReverseIterator::next() to iterate through the list in reverse order; do not use the iterator if ListReverseIterator::next() returns False. A list can be iterated in reverse order by calling ListReverseIterator::prev() instead of ListReverseIterator::next()

Example: ListIterator basic usage

```
my list $data = (1, "foo", 2);
my ListReverseIterator $it($data);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: 2
iter: "foo"
iter: 1
```

Note

- In general, the ListReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- ListReverseIterator is functionally equivalent to ListIterator, but the effect of ListReverseIterator::next() and ListReverseIterator::prev() are the opposite of ListIterator::next() and ListIterator::prev(); that is ListReverseIterator::next() will iterate through the list in reverse order, while ListReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for ListReverseIterator::first() and ListReverseIterator::last() are swapped in respect to ListIterator::first() and ListIterator::last().

See also

ListIterator

45.33.2 Member Function Documentation

45.33.2.1 Qore::ListReverseIterator::constructor (softlist *l*)

Creates the list iterator object.

Parameters

l	the list to iterate

Example:

```
my ListReverseIterator $li($l);
```

Note

the constructor's argument is softlist so that it can also accept NOTHING

45.33.2.2 Qore::ListReverseIterator::copy ( )

Creates a copy of the ListReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my ListReverseIterator $ni = $i.copy();
```

45.33.2.3 bool Qore::ListReverseIterator::first ( ) [virtual]

returns True if on the first element iterated with this iterator (ie the last element in the list)

Returns

True if on the first element iterated with this iterator (ie the last element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Reimplemented from Qore::ListIterator.

45.33.2.4 bool Qore::ListReverseIterator::last ( ) [virtual]

returns True if on the last element iterated with this iterator (ie the first element in the list)

Returns

True if on the last element iterated with this iterator (ie the first element in the list)

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Reimplemented from Qore::ListIterator.

45.33.2.5 bool Qore::ListReverseIterator::next ( ) [virtual]

Moves the current position to the previous element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the list if the list is not empty.

This method will return True again after it returns False once if the list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

ListReverseIterator::next() is the opposite of ListIterator::next(); it is functionally equivalent to ListIterator::prev(); ListReverseIterator::next() iterates through the list in reverse order

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::ListIterator.

45.33.2.6 bool Qore::ListReverseIterator::prev ( ) [virtual]

Moves the current position to the next element in the list; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the list if the list is not empty.

This method will return True again after it returns False once if list is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the list (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

ListReverseIterator::prev() is the opposite of ListIterator::prev(); it is functionally equivalent to ListIterator::next(); ListReverseIterator::prev() iterates through the list in forward order

Exceptions

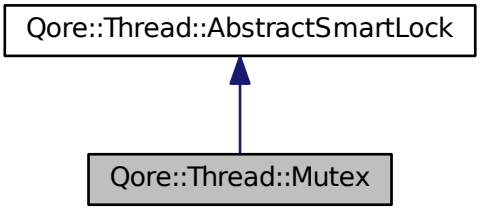
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::ListIterator.

45.34 Qore::Thread::Mutex Class Reference

A class providing an implementation for a simple thread lock.

Inheritance diagram for Qore::Thread::Mutex:



Public Member Functions

- constructor ()

  Creates the Mutex object.
- copy ()

  Creates a new Mutex object, not based on the original.
- destructor ()

  Destroys the object.
- nothing lock ()

  Locks the Mutex object; blocks if the lock is already held.
- int lock (timeout timeout_ms)

  Locks the Mutex object; blocks if the lock is already held.
- int trylock ()

  Acquires the lock only if it is not already held; returns 0 for success (lock acquired) or -1 if the call would block.
- nothing unlock ()

  Unlocks the Mutex object; wakes up one thread if any threads are blocked on this lock.

45.34.1 Detailed Description

A class providing an implementation for a simple thread lock.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

This class inherits AbstractSmartLock, so it can be used by Condition objects.

The Mutex class implements a mutual-exclusion lock for thread locking. Like all Qore thread primitives, objects of this class participate in deadlock detection and throw exceptions when threading errors occur (ex: unlocking a Mutex object locked by another thread, etc). See individual methods for more information on exceptions thrown.

See the AutoLock class for a class that assists in exception-safe Mutex locking.

Additionally, the on_exit statement can provide exception-safe unlocking at the lexical block level for Mutex objects as in the following example:

```
{
    $m.lock();
    on_exit
        $m.unlock();
```

```
    # ... when this block exits the lock will be released, even in the
    #     case of return statements or exceptions
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option

45.34.2 Member Function Documentation

45.34.2.1 Qore::Thread::Mutex::constructor ( )

Creates the Mutex object.

Example:

```
my Mutex $mutex();
```

45.34.2.2 Qore::Thread::Mutex::copy ( )

Creates a new Mutex object, not based on the original.

Example:

```
my Mutex $nm = $m.copy();
```

45.34.2.3 Qore::Thread::Mutex::destructor ( )

Destroys the object.

Note that it is a programming error to delete this object while other threads are blocked on it; in this case an exception is thrown in the deleting thread, and in each thread blocked on this object when it is deleted.

Example:

```
delete $mutex;
```

Exceptions

LOCK-ERROR	Object deleted while other threads blocked on it

45.34.2.4 nothing Qore::Thread::Mutex::lock ( )

Locks the Mutex object; blocks if the lock is already held.

To release the Mutex, use Mutex::unlock()

Example:

```
$mutex.lock();
```

Exceptions

LOCK-ERROR	lock called twice in the same thread, object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.34.2.5 int Qore::Thread::Mutex::lock (timeout *timeout_ms*)

Locks the Mutex object; blocks if the lock is already held.

An optional timeout value may be passed to this method, giving a time in milliseconds to wait for the lock to become free. Like all Qore functions and methods taking timeout values, a relative time value may be passed instead of an integer to make the timeout units clear

To release the Mutex, use Mutex::unlock()

Example:

```
if ($mutex.lock(1250ms))
    throw "TIMEOUT-ERROR", "lock acquisition timed out after 1.25s";
```

Parameters

timeout_ms	a timeout value to wait to acquire the lock; integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds)

Returns

returns -1 for error, 0 for success

Exceptions

LOCK-ERROR	lock called twice in the same thread, object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.34.2.6 int Qore::Thread::Mutex::trylock ( )

Acquires the lock only if it is not already held; returns 0 for success (lock acquired) or -1 if the call would block.

Returns

0 for success (lock acquired) or -1 if the call would block (lock not acquired)

Example:

```
my int $i = $mutex.trylock();
```

Exceptions

LOCK-ERROR	object deleted in another thread, etc
THREAD-DEADLOCK	a deadlock was detected while trying to acquire the lock

45.34.2.7 nothing Qore::Thread::Mutex::unlock ( )

Unlocks the Mutex object; wakes up one thread if any threads are blocked on this lock.

Example:

```
$mutex.unlock();
```

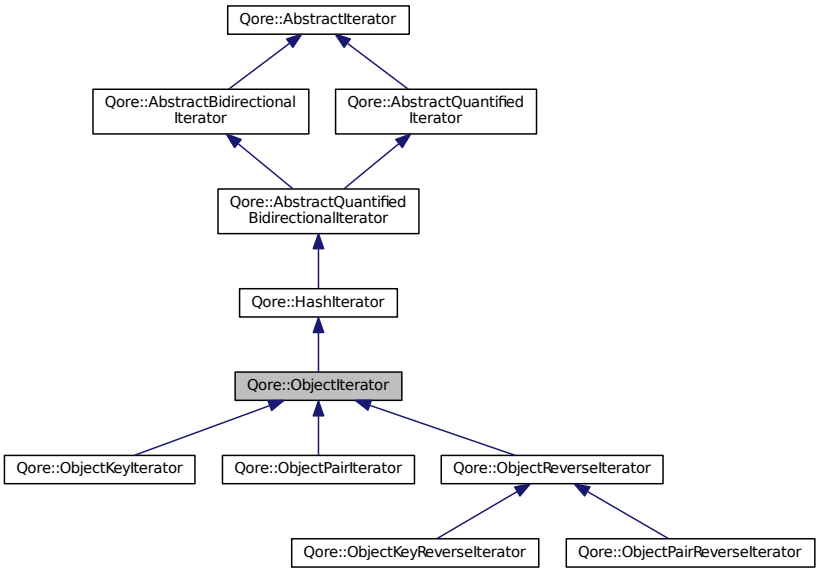
Exceptions

LOCK-ERROR	unlock called by a thread that does not own the lock or the lock is not locked, object deleted in another thread, etc

45.35 Qore::ObjectIterator Class Reference

This class a basic iterator class for objects.

Inheritance diagram for Qore::ObjectIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty object iterator object.
- copy ()

  Creates a copy of the ObjectIterator object, iterating the same object as the original and in the same position.

45.35.1 Detailed Description

This class a basic iterator class for objects.

Call ObjectIterator::next() to iterate through the object; do not use the iterator if ObjectIterator::next() returns False. An object can be iterated in reverse order by calling ObjectIterator::prev() instead of ObjectIterator::next()

Example: ObjectIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
```

```
    }
    constructor() {
        $.attr1 = 1;
        $.attr2 = now();
    }
}
my Class1 $o();

my ObjectIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: 1
iter: 2013-04-17 16:27:28 Wed +02:00 (CEST)
```

Note

- In general, the ObjectIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

ObjectReverseIterator

45.35.2 Member Function Documentation

45.35.2.1 Qore::ObjectIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectIterator $i($obj);
```

45.35.2.2 Qore::ObjectIterator::constructor ( )

Creates an empty object iterator object.

Example:

```
my *object $obj = get_object_or_nothing();
my ObjectIterator $i($obj);
```

45.35.2.3 Qore::ObjectIterator::copy ( )

Creates a copy of the ObjectIterator object, iterating the same object as the original and in the same position.

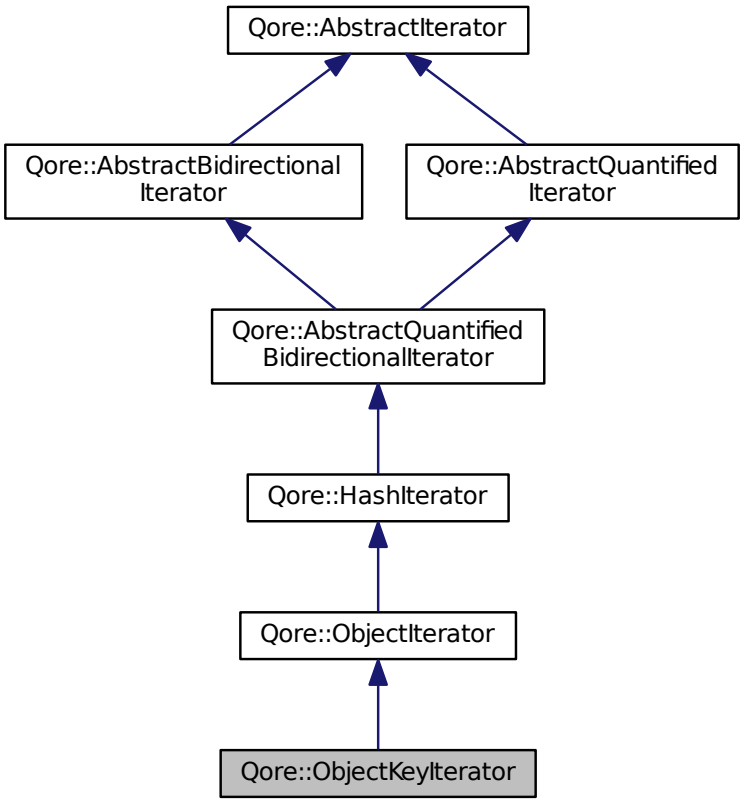
Example:

```
my ObjectIterator $ni = $i.copy();
```

45.36 Qore::ObjectKeyIterator Class Reference

This class an iterator class for objectes.

Inheritance diagram for Qore::ObjectKeyIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty object iterator object.
- copy ()

  Creates a copy of the ObjectKeyIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

45.36.1 Detailed Description

This class an iterator class for objectes.

Call ObjectKeyIterator::next() to iterate through the object; do not use the iterator if ObjectKeyIterator::next() returns False. A object can be iterated in reverse order by calling ObjectKeyIterator::prev() instead of ObjectKeyIterator::next()

Example: ObjectKeyIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
   }
   constructor() {
      $.attr1 = 1;
      $.attr2 = now();
   }
}
my Class1 $o();

my ObjectKeyIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: "attr1"
iter: "attr2"
```

Note

- In general, the ObjectKeyIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

ObjectKeyReverseIterator

45.36.2 Member Function Documentation

45.36.2.1 Qore::ObjectKeyIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectKeyIterator $i($h);
```

45.36.2.2 Qore::ObjectKeyIterator::constructor ( )

Creates an empty object iterator object.

Example:

```
my *object $o = get_object_or_nothing();
my ObjectKeyIterator $i($o);
```

45.36.2.3 Qore::ObjectKeyIterator::copy ( )

Creates a copy of the ObjectKeyIterator object, iterating the same object as the original and in the same position.

Example:

```
my ObjectKeyIterator $ni = $i.copy();
```

45.36.2.4 string Qore::ObjectKeyIterator::getValue ( ) [virtual]

returns the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

the current key value or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
foreach my string $key in ($object.keyIterator())
    printf("key: %s\n", $key);
```

Exceptions

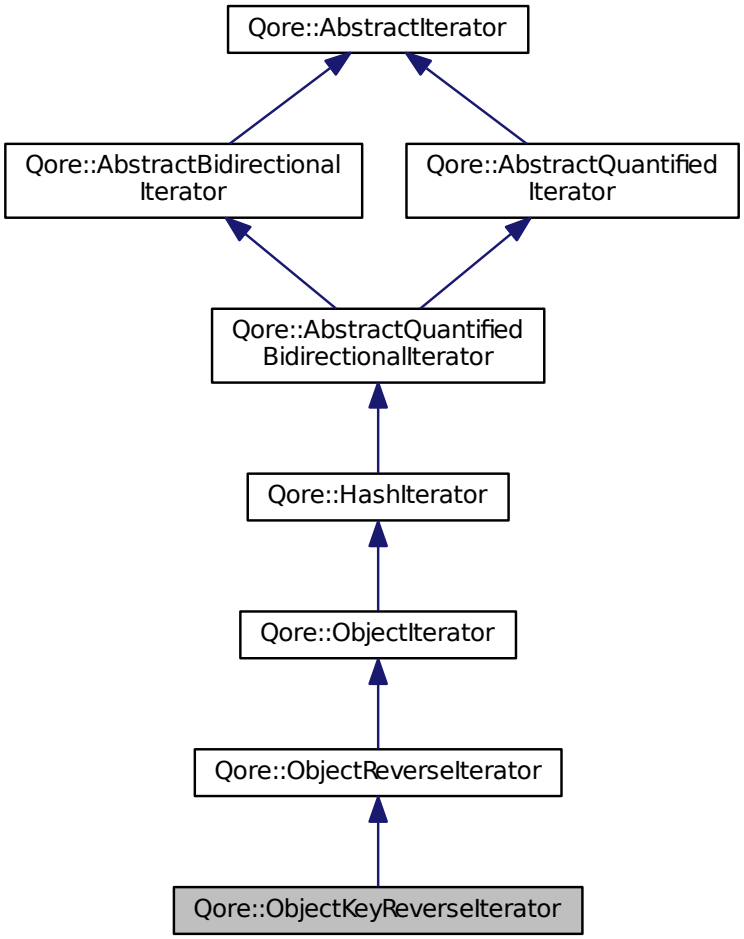
INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.37 Qore::ObjectKeyReverseIterator Class Reference

This class an iterator class for objects.

Inheritance diagram for Qore::ObjectKeyReverseIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty iterator object.
- copy ()

  Creates a copy of the ObjectKeyReverseIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

45.37.1 Detailed Description

This class an iterator class for objects.

Call ObjectKeyReverseIterator::next() to iterate through the object in reverse order; do not use the iterator if ObjectKeyReverseIterator::next() returns False. A object can be iterated in reverse order by calling ObjectKeyReverseIterator::prev() instead of ObjectKeyReverseIterator::next()

Example: ObjectKeyReverseIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
   }
   constructor() {
      $.attr1 = 1;
      $.attr2 = now();
   }
}
my Class1 $o();

my ObjectKeyReverseIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: "attr2"
iter: "attr1"
```

Note

- In general, the ObjectKeyReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- ObjectKeyReverseIterator is functionally equivalent to ObjectKeyIterator, but the effect of ObjectKeyReverseIterator::next() and ObjectKeyReverseIterator::prev() are the opposite of ObjectKeyIterator::next() and ObjectKeyIterator::prev(); that is ObjectKeyReverseIterator::next() will iterate through the object in reverse order, while ObjectKeyReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for ObjectKeyReverseIterator::first() and ObjectKeyReverseIterator::last() are swapped in respect to ObjectKeyIterator::first() and ObjectKeyIterator::last().

See also

ObjectKeyIterator

45.37.2 Member Function Documentation

45.37.2.1 Qore::ObjectKeyReverseIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectKeyReverseIterator $i($obj);
```

45.37.2.2 Qore::ObjectKeyReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *object $obj = get_object();
my ObjectKeyReverseIterator $i($obj);
```

45.37.2.3 Qore::ObjectKeyReverseIterator::copy ( )

Creates a copy of the ObjectKeyReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my ObjectKeyReverseIterator $ni = $i.copy();
```

45.37.2.4 string Qore::ObjectKeyReverseIterator::getValue ( ) [virtual]

returns the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current key value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
my ObjectKeyReverseIterator $i($obj);
while ($i.next())
    printf("key: %s\n", $i.getValue());
```

Exceptions

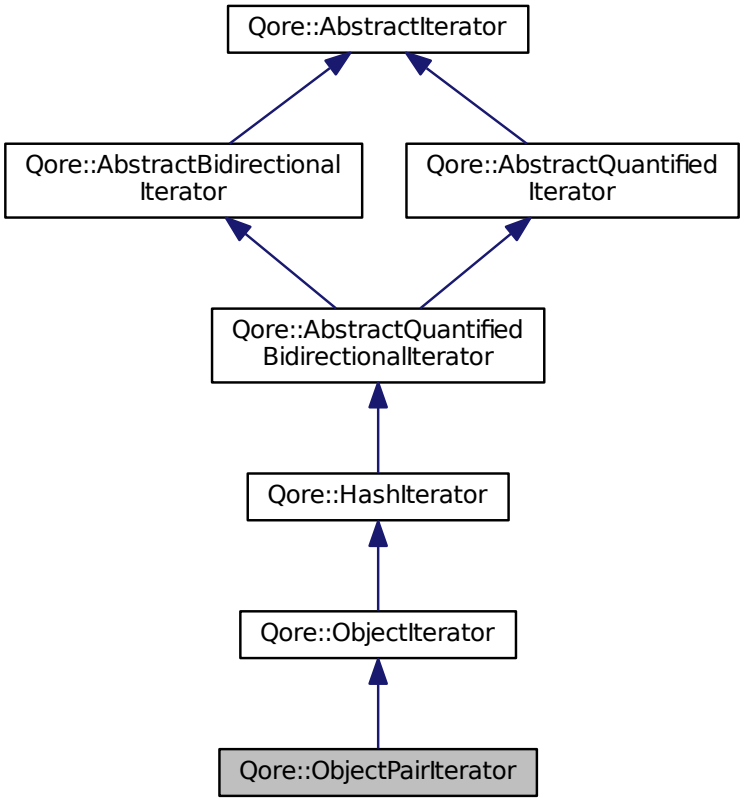
INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.38 Qore::ObjectPairIterator Class Reference

This class an iterator class for objects.

Inheritance diagram for Qore::ObjectPairIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty object iterator object.
- copy ()

  Creates a copy of the ObjectPairIterator object, iterating the same object as the original and in the same position.
- hash getValue ()

  returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

45.38.1 Detailed Description

This class an iterator class for objects.

Call ObjectPairIterator::next() to iterate through the object; do not use the iterator if ObjectPairIterator::next() returns False. A object can be iterated in reverse order by calling ObjectPairIterator::prev() instead of ObjectPairIterator::next()

Example: ObjectPairIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
   }
   constructor() {
      $.attr1 = 1;
      $.attr2 = now();
   }
}
my Class1 $o();

my ObjectPairIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: hash: (key : "attr1", value : 1)
iter: hash: (key : "attr2", value : 2013-04-17 16:37:26 Wed +02:00 (CEST))
```

Note

- In general, the ObjectPairIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.

See also

ObjectPairReverseIterator

45.38.2 Member Function Documentation

45.38.2.1 Qore::ObjectPairIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectPairIterator $i($obj);
```

45.38.2.2 Qore::ObjectPairIterator::constructor ( )

Creates an empty object iterator object.

Example:

```
my *object $obj = get_object_or_nothing();
my ObjectPairIterator $i($obj);
```

45.38.2.3 Qore::ObjectPairIterator::copy ( )

Creates a copy of the ObjectPairIterator object, iterating the same object as the original and in the same position.

Example:

```
my ObjectPairIterator $ni = $i.copy();
```

45.38.2.4 hash Qore::ObjectPairIterator::getValue ( ) [virtual]

returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
map printf("%s: %y\n", $1.key, $1.value), $object.pairIterator();
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

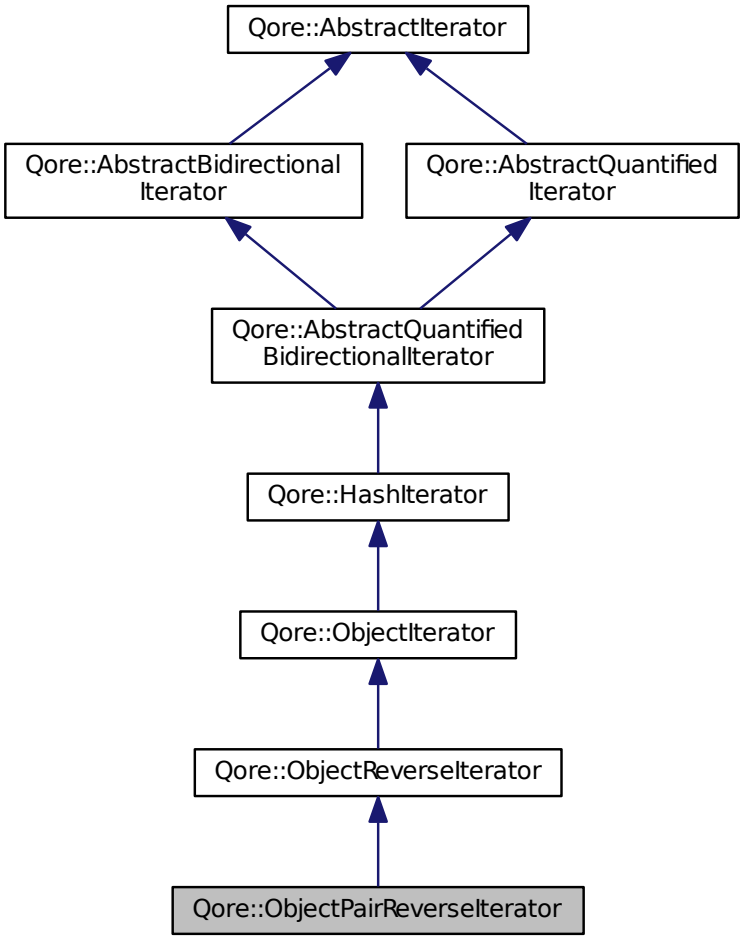
Qore 0.8.6.2

Reimplemented from Qore::HashIterator.

45.39 Qore::ObjectPairReverseIterator Class Reference

This class an iterator class for objects.

Inheritance diagram for Qore::ObjectPairReverseIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty iterator object.
- copy ()

  Creates a copy of the ObjectPairReverseIterator object, iterating the same object as the original and in the same position.
- string getValue ()

  returns a hash with the current key and value (a hash with 2 keys: "`key`" and "`value`") or throws an `INVALID-ITERATOR` exception if the iterator is invalid

45.39.1 Detailed Description

This class an iterator class for objects.

Call ObjectPairReverseIterator::next() to iterate through the object in reverse order; do not use the iterator if ObjectPairReverseIterator::next() returns False. A object can be iterated in reverse order by calling ObjectPairReverseIterator::prev() instead of ObjectPairReverseIterator::next()

Example: ObjectPairReverseIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
   }
   constructor() {
      $.attr1 = 1;
      $.attr2 = now();
   }
}
my Class1 $o();

my ObjectPairReverseIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: hash: (key : "attr2", value : 2013-04-17 16:38:31 Wed +02:00 (CEST))
iter: hash: (key : "attr1", value : 1)
```

Note

- In general, the ObjectPairReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- ObjectPairReverseIterator is functionally equivalent to ObjectPairIterator, but the effect of ObjectPairReverseIterator::next() and ObjectPairReverseIterator::prev() are the opposite of ObjectPairIterator::next() and ObjectPairIterator::prev(); that is ObjectPairReverseIterator::next() will iterate through the object in reverse order, while ObjectPairReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for ObjectPairReverseIterator::first() and ObjectPairReverseIterator::last() are swapped in respect to ObjectPairIterator::first() and ObjectPairIterator::last().

See also

ObjectPairIterator

45.39.2 Member Function Documentation

45.39.2.1 Qore::ObjectPairReverseIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectPairReverseIterator $i($obj);
```

45.39.2.2 Qore::ObjectPairReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *object $obj = get_object_or_nothing();
my ObjectPairReverseIterator $i($obj);
```

45.39.2.3 Qore::ObjectPairReverseIterator::copy ( )

Creates a copy of the ObjectPairReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my ObjectPairReverseIterator $ni = $i.copy();
```

45.39.2.4 string Qore::ObjectPairReverseIterator::getValue ( ) [virtual]

returns a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

a hash with the current key and value (a hash with 2 keys: "key" and "value") or throws an INVALID-ITERATOR exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
foreach my hash $h in (new ObjectPairReverseIterator($obj))
    printf("%s: %y\n", $h.key, $h.value);
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Since

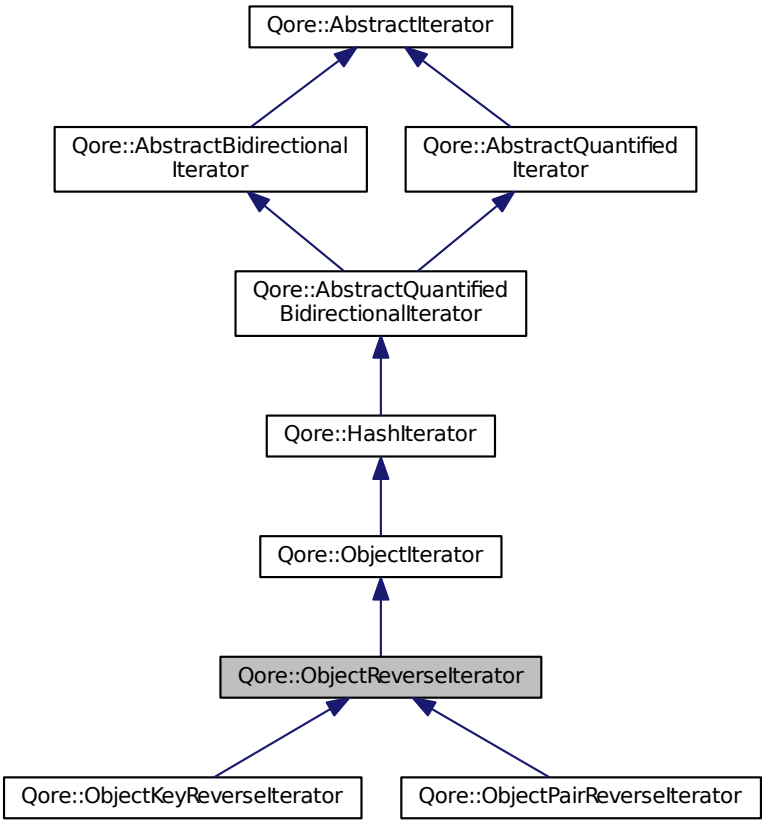
Qore 0.8.6.2

Reimplemented from Qore::HashIterator.

45.40 Qore::ObjectReverseIterator Class Reference

This class an iterator class for objects.

Inheritance diagram for Qore::ObjectReverseIterator:



Public Member Functions

- constructor (object o)

  Creates the object iterator object.
- constructor ()

  Creates an empty iterator object.
- copy ()

  Creates a copy of the ObjectReverseIterator object, iterating the same object as the original and in the same position.
- bool first ()

  returns True if on the last element of the object
- bool last ()

returns True if on the first element of the object

- bool next ()

  Moves the current position to the previous element in the object; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the object if the object is not empty.

- bool prev ()

  Moves the current position to the next element in the object; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the object if the object is not empty.

45.40.1 Detailed Description

This class an iterator class for objects.

Call ObjectReverseIterator::next() to iterate through the object in reverse order; do not use the iterator if ObjectReverseIterator::next() returns False. An object can be iterated in reverse order by calling ObjectReverseIterator::prev() instead of ObjectReverseIterator::next()

Example: ObjectReverseIterator basic usage

```
class Class1 {
   public {
      int $.attr1;
      date $.attr2;
   }
   constructor() {
      $.attr1 = 1;
      $.attr2 = now();
   }
}
my Class1 $o();

my ObjectReverseIterator $it($o);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: 2013-04-17 16:31:03 Wed +02:00 (CEST)
iter: 1
```

Note

- In general, the ObjectReverseIterator class is not designed to be accessed from multiple threads; it was created without locking for fast and efficient use when used from a single thread. For methods that would be unsafe to use in another thread, any use of such methods in threads other than the thread where the constructor was called will cause an `ITERATOR-THREAD-ERROR` to be thrown.
- ObjectReverseIterator is functionally equivalent to ObjectIterator, but the effect of ObjectReverseIterator::next() and ObjectReverseIterator::prev() are the opposite of ObjectIterator::next() and ObjectIterator::prev(); that is ObjectReverseIterator::next() will iterate through the object in reverse order, while ObjectReverseIterator::prev() iterates in forward order. Additionally the meanings of the return values for ObjectReverseIterator::first() and ObjectReverseIterator::last() are swapped in respect to ObjectIterator::first() and ObjectIterator::last().

See also

ObjectIterator

45.40.2 Member Function Documentation

45.40.2.1 Qore::ObjectReverseIterator::constructor (object *o*)

Creates the object iterator object.

Parameters

o	the object to iterate

Example:

```
my ObjectReverseIterator $i($obj);
```

45.40.2.2 Qore::ObjectReverseIterator::constructor ( )

Creates an empty iterator object.

Example:

```
my *object $obj = get_object_or_nothing();
my ObjectReverseIterator $i($obj);
```

45.40.2.3 Qore::ObjectReverseIterator::copy ( )

Creates a copy of the ObjectReverseIterator object, iterating the same object as the original and in the same position.

Example:

```
my ObjectReverseIterator $ni = $i.copy();
```

45.40.2.4 bool Qore::ObjectReverseIterator::first ( ) [virtual]

returns True if on the last element of the object

Returns

True if on the last element of the object

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.first())
        printf("START:\n");
}
```

Reimplemented from Qore::HashIterator.

45.40.2.5 bool Qore::ObjectReverseIterator::last ( ) [virtual]

returns True if on the first element of the object

Returns

True if on the first element of the object

Code Flags:

CONSTANT

Example:

```
while ($i.next()) {
    if ($i.last())
        printf("END.\n");
}
```

Reimplemented from Qore::HashIterator.

45.40.2.6 bool Qore::ObjectReverseIterator::next ( ) [virtual]

Moves the current position to the previous element in the object; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the last element in the object if the object is not empty.

This method will return True again after it returns False once if the object is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the object (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.prev()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

ObjectReverseIterator::next() is the opposite of ObjectIterator::next(); it is functionally equivalent to ObjectIterator::prev(); ObjectReverseIterator::next() iterates through the object in reverse order

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.40.2.7 bool Qore::ObjectReverseIterator::prev ( ) [virtual]

Moves the current position to the next element in the object; returns False if there are no more elements; if the iterator is not pointing at a valid element before this call, the iterator will be positioned on the first element in the object if the object is not empty.

This method will return True again after it returns False once if object is not empty, otherwise it will always return False. The iterator object should not be used after this method returns False

Returns

False if there are no more elements in the object (in which case the iterator object is invalid and should not be used); True if successful (meaning that the iterator object is valid)

Example:

```
while ($i.next()) {
    printf(" + %y\n", $i.getValue());
}
```

Note

ObjectReverseIterator::prev() is the opposite of ObjectIterator::prev(); it is functionally equivalent to ObjectIterator::next(); ObjectReverseIterator::prev() iterates through the object in forward order

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Reimplemented from Qore::HashIterator.

45.41 Qore::Program Class Reference

Program objects allow Qore programs to support subprograms with the option to restrict capabilities, for example, to support user-defined logic for application actions.

Public Member Functions

- any callFunction (string name,...)

  Calls a function in the program object and returns the return value.
- any callFunctionArgs (string name, __7_ softlist vargs)

  Calls a function in the program object giving the arguments to the function as a list and returns the return value.
- constructor (softint po=PO_DEFAULT)

  Creates the program object and optionally sets program capabilities (parse options)
- copy ()

  Throws an exception to prevent objects of this class from being copied.
- nothing define (string def, any val)

  Sets a parse define for the current Program.
- destructor ()

  Waits for all threads to finish executing, then deletes all global variables, dereferences the internal Program object and deletes the Qore object.
- nothing disableParseOptions (softint opt)

  Removes the given parse options to the current parse option mask.
- bool existsFunction (string name)

  Checks if a user function exists in the program object.
- any getDefine (string def)

  Retrieves the value of the given parse define in the current Program.
- any getGlobalVariable (string varname, __7_ reference rexists)

  Returns a the value of the global variable identified by the first string argument.
- int getParseOptions ()

  Returns the current binary-or'ed parse option mask for the Program object.
- __7_ string getScriptDir ()

Returns the current script directory as a string or NOTHING if not set.

- __7_ string getScriptName ()

  Returns the current script name as a string or NOTHING if not set.

- __7_ string getScriptPath ()

  Returns the current script directory and filename if known, otherwise returns NOTHING.

- TimeZone getTimeZone ()

  Returns the default local time zone for the object.

- list getUserFunctionList ()

  Returns a list of strings of all user functions defined in the program object.

- nothing importClass (string cls)

  Imports a class into the program object's space; any calls to the imported class's code will run with the parent Program object's permissions.

- nothing importFunction (string func)

  Imports a function into the program object's space; any calls to the imported function will run with the parent Program object's permissions.

- nothing importFunction (string func, string new_name)

  Imports a function into the program object's space and gives it a new name; any calls to the imported function will run with the parent Program object's permissions.

- nothing importGlobalVariable (string varname, bool readonly=False)

  Imports a global variable into the program object's space.

- bool isDefined (string def)

  Returns True if the given parse define is defined in the current Program (does not have to have a value defined to return True), False if not.

- loadModule (string name)

  Loads a Qore module into the Program object at run-time.

- nothing lockOptions ()

  Locks parse options so that they cannot be changed.

- __7_ hash parse (string code, string label, __7_ softint warning_mask, __7_ string source, __7_ softint offset, softbool format_label=True)

  Parses the string argument and adds the code to the Program object.

- nothing parseCommit ()

  Commits and pending code processed with Program::parsePending() to the Program object after resolving all outstanding references in the pending code.

- __7_ hash parseCommit (int warning_mask)

  Commits and pending code processed with Program::parsePending() to the Program object after resolving all outstanding references in the pending code.

- __7_ hash parsePending (string code, string label, __7_ softint warning_mask, __7_ string source, __7_ softint offset, softbool format_label=True)

  Parses the text passed to pending lists in the Program object; does not resolve all references or commit the code to the Program object.

- nothing parseRollback ()

  Rolls back any pending code processed with Program::parsePending() that has not yet been committed to the Program object with Program::parseCommit()

- nothing replaceParseOptions (softint opt)

  Replaces the parse options for the Program object.

- any run ()

  Runs the program and optionally returns a value if the top-level code exits with a return statement.

- nothing setParseOptions (softint opt=PO_DEFAULT)

  Sets parse options in the parse option mask for the Program object.

- nothing setScriptPath (__7_ string path)

  Sets (or clears) the script path (directory and filename) for the object.

- nothing setTimeZone (TimeZone zone)

Sets the default local time zone for the object.

- nothing setTimeZoneRegion (string region)

  Sets the default local time zone for the object from a path to a zoneinfo time zone region file.

- nothing setTimeZoneUTCOffset (softint seconds_east)

  Sets the default time zone for the Program object based on the number of seconds east of UTC; for zones west of UTC, use negative numbers.

- nothing undefine (string def)

  Unsets a parse define for the current Program.

45.41.1 Detailed Description

Program objects allow Qore programs to support subprograms with the option to restrict capabilities, for example, to support user-defined logic for application actions.

Parsing in Qore happens in two steps; first all code is parsed to pending data structures, and then in the second stage, all references are resolved, and, if there are no errors, then all changes are committed to the Program object. Note that all parse actions (Program::parse(), Program::parsePending(), Program::parseCommit(), and Program::parseRollback()) are atomic; there is a thread lock on each Program object to ensure atomicity, and if any parse errors occur in any stage of parsing, any pending changes to the Program object are automatically rolled back. However parse actions that affect only one stage of the two stages of parsing (Program::parsePending(), Program::parseCommit() and Program::parseRollback()) are atomic within themselves, but not between calls, so one thread may inadvertently commit changes to a Program object if two or more threads are trying to perform transaction-safe two-stage parsing on a Program object without explicit user locking.

Parse option constants can be used to limit the capabilities of a Program object. These options should be binary-OR'ed together and passed to the Program object's constructor. Also see qore Executable Command-Line Processing for equivalent command-line options, and Parse Directives for equivalent parse directives.

Note that a program can provide controlled access to functionality otherwise restricted by parse options by exporting a custom API into the child program object using either the Program::importFunction() or Program::importGlobalVariable() method. This is possible because code (functions or object methods) imported into and called from a subprogram will run in the parent's space and therefore with the parent's capabilities.

Classes, constants, namespaces, functions, and global variables are only inherited into child Program objects if they are declared public and no parse options prohibit it.

Symbols can also be imported into Program objects singly using methods such as Program::importClass() and Program::importGlobalVariable(), etc.

45.41.2 Member Function Documentation

45.41.2.1 any Qore::Program::callFunction (string *name*, ...)

Calls a function in the program object and returns the return value.

The function runs with the permissions of the Program object containing the function.

Parameters

name	The name of the function to call
...	The remaining arguments passed to the method are passed to the function to be called

Returns

Depends on the function being called

Example:

```
my any $result = $pgm.callFunction("func_name", $arg1, $arg2);
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow this builtin function to be called
NO-FUNCTION	The function does not exist
ENCODING-CONVERSION-ERROR	the function name could not be converted to the default character encoding

Note

The function called could also cause other exceptions to be thrown

45.41.2.2 any Qore::Program::callFunctionArgs (**string** *name*, __7_ **softlist** *vargs*)

Calls a function in the program object giving the arguments to the function as a list and returns the return value.

The function runs with the permissions of the Program object containing the function.

Parameters

name	The name of the function to call
vargs	The arguments to the function to be called

Returns

Depends on the function being called

Example:

```
my any $result = $pgm.callFunctionArgs("func_name", ($arg1, $arg2));
```

Exceptions

INVALID-FUNCTION-ACCESS	Parse options do not allow this builtin function to be called
NO-FUNCTION	The function does not exist
ENCODING-CONVERSION-ERROR	the function name could not be converted to the default character encoding

Note

The function called could also cause other exceptions to be thrown

45.41.2.3 Qore::Program::constructor (softint *po* = **PO_DEFAULT**)

Creates the program object and optionally sets program capabilities (parse options)

Note that if PO_NO_CHILD_PO_RESTRICTIONS is not set in the parent Program when the new Program object is created, then the created Program object will have the parent's parse options added to its parse options as given by the argument to the constructor.

In other words, if PO_NO_CHILD_PO_RESTRICTIONS is not set, it's not possible to create a child Program object with fewer restrictions than the parent Program object (any attempt to do so will be silently ignored).

However, if PO_NO_CHILD_PO_RESTRICTIONS is set in the parent Program object, then the parse option argument to the constructor will be applied literally to the child object.

Restrictions:

Qore::PO_NO_EMBEDDED_LOGIC

Parameters

po	A binary OR'ed product of parse options

Example:

```
my Program $pgm();
```

Exceptions

PROGRAM-OPTION-ERROR	invalid parse options used

45.41.2.4 Qore::Program::copy ( )

Throws an exception to prevent objects of this class from being copied.

Exceptions

PROGRAM-COPY-ERROR	copying Program objects is currently unsupported

45.41.2.5 nothing Qore::Program::define (string *def*, any *val*)

Sets a parse define for the current Program.

Parameters

def	The parse define to assign
val	The value to assign to the define

Example:

```
$pgm.define("PRODUCTION", True);
```

45.41.2.6 nothing Qore::Program::disableParseOptions (softint *opt*)

Removes the given parse options to the current parse option mask.

An `OPTIONS-LOCKED` exception is thrown if parse options have been locked (for example with Program::lockOptions())

Parameters

opt	A single parse option or binary-or'ed combination of parse options to unset in the parse option mask for the object; any bit set in this argument will be unset (ie zeroed out or combined with inverse binary and) in the Program's parse option mask

Example:

```
# allow threading and GUI operations
$pgm.disableParseOptions(PO_NO_THREADS | PO_NO_GUI);
```

Exceptions

OPTIONS-LOCKED	Parse options have been locked and cannot be changed
PROGRAM-OPTION-ERROR	invalid parse options used

See also

Program::setParseOptions() for a reciprocal method that enables parse options; also see Program::replaceParseOptions()

45.41.2.7 bool Qore::Program::existsFunction (string *name*)

Checks if a user function exists in the program object.

Code Flags:

RET_VALUE_ONLY

Parameters

name	The name of the function to check

Returns

True if the function exists, False if not

Example:

```
if ($pgm.existsFunction("my_func"))
    printf("my_func() exists in the Program\n");
```

Exceptions

ENCODING-CONVERSION-ERROR	the function name could not be converted to the default character encoding

45.41.2.8 any Qore::Program::getDefine (string *def*)

Retrieves the value of the given parse define in the current Program.

Returns

the value of the given parse define in the current Program

Example:

```
my any $val = $pgm.getDefine("PRODUCTION");
```

Note

A parse define may be defined with no value; use Program::isDefined to check if a parse define is actually defined or not

45.41.2.9 any Qore::Program::getGlobalVariable (string *varname*, __7_ reference *rexists*)

Returns a the value of the global variable identified by the first string argument.

Parameters

varname	The string name of the global variable to find, not including the leading "$" character
rexists	An lvalue reference to a bool to determine if the global variable exists (because this method could return NOTHING when the variable exists as well as when it does not)

Returns

the value of the global variable identified by the first string argument giving the name of the variable (without any leading "$" symbol)

Example:

```
my bool $exists;
my any $val = $pgm.getGlobalVariable("error_count", \$exists);
```

45.41.2.10 int Qore::Program::getParseOptions ( )

Returns the current binary-or'ed parse option mask for the Program object.

Returns

the current binary-or'ed parse option mask for the Program object

Example:

```
my int $mask = $pgm.getParseOptions();
```

45.41.2.11 __7_ string Qore::Program::getScriptDir ( )

Returns the current script directory as a string or NOTHING if not set.

Gets the script directory set with Program::setScriptPath(). This is the same value that will be returned in the Qore program code with the get_script_dir() function if called from within the Program.

Returns

Returns the current script directory as a string or NOTHING if not set; if a string is returned, it should normally include the trailing directory separator character ("/" on UNIX, "\\" on Windows)

Example:

```
my *string $dir = $pgm.getScriptDir();
```

45.41.2.12 __7_ string Qore::Program::getScriptName ( )

Returns the current script name as a string or NOTHING if not set.

Gets the script filename set with Program::setScriptPath(). This is the same value that will be returned in the Qore program code with the get_script_name() function if called from within the Program.

Returns

the current script name as a string or NOTHING if not set

Example:

```
my *string $name = $pgm.getScriptName();
```

45.41.2.13 __7_ string Qore::Program::getScriptPath ( )

Returns the current script directory and filename if known, otherwise returns NOTHING.

Gets the script directory and filename set with Program::setScriptPath(). This is the same value that will be returned in the Qore program code with the get_script_path() function if called from within the Program.

Returns

the current script directory and filename if known, otherwise returns NOTHING

Example:

```
my *string $path = $pgm.getScriptPath();
```

45.41.2.14 TimeZone Qore::Program::getTimeZone ( )

Returns the default local time zone for the object.

Returns

the default local time zone for the object

Example:

```
my TimeZone $tz = $pgm.getTimeZone();
```

45.41.2.15 list Qore::Program::getUserFunctionList ( )

Returns a list of strings of all user functions defined in the program object.

Returns

a list of strings of all user functions defined in the program object

Example:

```
my list $l = $pgm.getUserFunctionList();
```

45.41.2.16 nothing Qore::Program::importClass (string *cls*)

Imports a class into the program object's space; any calls to the imported class's code will run with the parent Program object's permissions.

This allows a user-defined API with greater capabilities than the embedded Program object to be imported into the embedded code.

Parameters

cls	the name of the class to import into the Program object

Example:

```
$pgm.importClass("MyNamespace::MyClass");
```

Exceptions

CLASS-IMPORT-ERROR	Cannot import a class into the same Program object; class or namespace with this name already exists or the source class does not exist
ENCODING-CONVERSION-ERROR	the class name could not be converted to the default character encoding

Since

Qore 0.8.7

45.41.2.17 nothing Qore::Program::importFunction (string *func*)

Imports a function into the program object's space; any calls to the imported function will run with the parent Program object's permissions.

This allows a user-defined API with greater capabilities than the embedded Program object to be imported into the embedded code.

Parameters

func	the name of the function to import into the Program object

Example:

```
$pgm.importFunction("log");
```

Exceptions

FUNCTION-IMPORT-ERROR	Cannot import a function into the same Program object; function with this name already exists
PROGRAM-IMPORTFUNCTION-NO-FUNCTION	The function does not exist
ENCODING-CONVERSION-ERROR	the function name could not be converted to the default character encoding

Since

Qore 0.8.4 the function name can include a namespace path (ex `"Namespace::func"`)

45.41.2.18 nothing Qore::Program::importFunction (string *func*, string *new_name*)

Imports a function into the program object's space and gives it a new name; any calls to the imported function will run with the parent Program object's permissions.

This allows a user-defined API with greater capabilities than the embedded Program object to be imported into the embedded code.

Parameters

func	the name of the function to import into the Program object
new_name	the name of the function as it will be known in the Program object once imported

Example:

```
$pgm.importFunction("service_log", "log");
```

Exceptions

FUNCTION-IMPORT-ERROR	Cannot import a function into the same Program object; function with this name already exists; target namespace does not exist
PROGRAM-IMPORTFUNCTION-NO-FUNCTION	The function does not exist
ENCODING-CONVERSION-ERROR	the function name could not be converted to the default character encoding

Since

Qore 0.8.4 the function name can include a namespace path (ex "`Namespace::func`")

45.41.2.19 nothing Qore::Program::importGlobalVariable (string *varname*, bool *readonly* = False)

Imports a global variable into the program object's space.

If the variable is an object, then any methods called from the subprogram will run in the parent's space

Parameters

varname	The name of the global variable without the "`$`"
readonly	If this argument is True, then the variable will be imported read-only, and cannot be changed by the subprogram (note that if the imported

Example:

```
$pgm.importGlobalVariable("var");
```

Exceptions

PROGRAM-IMPORTGLOBALVARIABLE-EXCEPTION	The global variable does not exist in the source program, or the target variable already exists
ENCODING-CONVERSION-ERROR	the global variablt name could not be converted to the default character encoding

45.41.2.20 bool Qore::Program::isDefined (string *def*)

Returns True if the given parse define is defined in the current Program (does not have to have a value defined to return True), False if not.

Parameters

def	The name of the define to check

Returns

True if the given parse define is defined in the current Program (does not have to have a value defined to return True), False if not

Example:

```
my bool $b = $pgm.isDefined("PRODUCTION");
```

45.41.2.21 Qore::Program::loadModule (string *name*)

Loads a Qore module into the Program object at run-time.

If a feature with the same name already exists, then this feature's code is imported into the current Program object if necessary and no further action is taken.

Note that modules providing objects resolved at parse time (classes, constants, functions, etc) must be loaded prior to parsing.

Restrictions:

Qore::PO_NO_MODULES

Parameters

name	either a feature name (a module will be searched with this feature name) or a path to a module to load

Example:

```
$pgm.loadModule("mysql");
```

Exceptions

LOAD-MODULE-ERROR	module cannot be loaded: API incompatibility, module defines symbols already defined in the target object, etc

See also

getModuleHash(), getFeatureList()

Since

Qore 0.8.7

45.41.2.22 nothing Qore::Program::lockOptions ( )

Locks parse options so that they cannot be changed.

Example:

```
$pgm.lockOptions();
```

45.41.2.23 __7_ hash Qore::Program::parse (string *code,* string *label,* __7_ softint *warning_mask,* __7_ string *source,* __7_ softint *offset,* softbool *format_label* = True)

Parses the string argument and adds the code to the Program object.

This method causes both stages of parsing to be executed; if this method is successful, then the code parsed is committed to the Program object. This method is equivalent to calling Program::parsePending() and Program::parseCommit() in one atomic call.

If an exception occurs in this method, all pending code is backed out, not just code parsed by this method (for example, in case uncommitted code added by Program::parsePending() also exists in the Program object before calling this method).

Restrictions:

Qore::PO_NO_EMBEDDED_LOGIC

Parameters

code	The code to parse into the Program object
label	The label for the code; this label will be given if any parse or run-time errors are raised for the code given
warning_mask	An optional warning mask; see Warning Constants for values to combine by binary-or; if this argument is 0 or not given then no warnings will be checked or issued and the return value will always be NOTHING
source	An optional source file name for the code being parsed; this is useful if sections of a file are parsed
offset	An optional line offset for use with the *source* parameter; this gives the line offset in the file to the code being parsed
format_label	If this argument is True, then the label is formatted as "<run-time-loaded : label>"; if False, then it is passed as-is

Returns

If warning included in the warning mask are raised during parsing, this method will return an exception hash with warning information, otherwise NOTHING is returned

Example:

```
my *hash $wh = $pgm.parse($code, "label", WARN_DEFAULT);
while ($wh) {
    printf("warning: %s:%d: %s: %s\n", $wh.file, $wh.line, $wh.err, $wh.desc);
    $wh = $wh.next;
}
```

Note

This method could throw many parse exceptions which are not enumerated here; any parse errors will result in an appropriate exception.

See also

- Qore::Program::parsePending()
- Qore::Program::parseCommit()
- Qore::parse()

Since

Qore 0.8.7 the *source*, *offset*, and *format_label* arguments were added

45.41.2.24 nothing Qore::Program::parseCommit ( )

Commits and pending code processed with Program::parsePending() to the Program object after resolving all outstanding references in the pending code.

An exception in this method causes all pending code to be rolled back immediately.

Example:

```
$pgm.parseCommit();
```

Note

This method could throw many parse exceptions related to resolving references which are not enumerated here; any parse errors will result in an appropriate exception.

See also

- Qore::Program::parse()
- Qore::Program::parsePending()
- Qore::Program::parseRollback()
- Qore::parse()

45.41.2.25 __7_ hash Qore::Program::parseCommit (int *warning_mask*)

Commits and pending code processed with Program::parsePending() to the Program object after resolving all outstanding references in the pending code.

An exception in this method causes all pending code to be rolled back immediately.

Example:

```
my *hash $wh = $pgm.parseCommit(WARN_DEFAULT);
while (exists $wh) {
    printf("warning: %s:%d: %s: %s\n", $wh.file, $wh.line, $wh.err, $wh.desc);
    $wh = $wh.next;
}
```

Note

This method could throw many parse exceptions related to resolving references which are not enumerated here; any parse errors will result in an appropriate exception.

See also

- Qore::Program::parse()
- Qore::Program::parsePending()
- Qore::Program::parseRollback()
- Qore::parse()

45.41.2.26 __7_ hash Qore::Program::parsePending (string *code,* string *label,* __7_ softint *warning_mask,* __7_ string *source,* __7_ softint *offset,* softbool *format_label* = True)

Parses the text passed to pending lists in the Program object; does not resolve all references or commit the code to the Program object.

References are resolved in the Program::parseCommit() method.

Program::parseCommit() must be called to resolve all references and commit the code to the Program object; until Program::parseCommit() is called, none of the code parsed by this method will be available for execution in the Program object.

If an exception occurs in this method, all pending code is backed out, not just code parsed by this method.

Restrictions:

Qore::PO_NO_EMBEDDED_LOGIC

Parameters

code	The code to parse into the Program object
label	The label for the code; this label will be given if any parse or run-time errors are raised for the code given
warning_mask	An optional warning mask; see Warning Constants for values to combine by binary-or; if this argument is 0 or not given then no warnings will be checked or issued and the return value will always be NOTHING
source	An optional source file name for the code being parsed; this is useful if sections of a file are parsed
offset	An optional line offset for use with the *source* parameter; this gives the line offset in the file to the code being parsed
format_label	If this argument is True, then the label is formatted as "<run-time-loaded : label>"; if False, then it is passed as-is

Returns

If warning included in the warning mask are raised during parsing, this method will return an exception hash with warning information, otherwise NOTHING is returned

Example:

```
my *hash $wh = $pgm.parsePending($code, "label", WARN_DEFAULT);
while ($wh) {
    printf("warning: %s:%d: %s: %s\n", $wh.file, $wh.line, $wh.err, $wh.desc);
    $wh = $wh.next;
}
$pgm.parseCommit();
```

Note

This method could throw many parse exceptions which are not enumerated here; any parse errors will result in an appropriate exception.

See also

- Qore::Program::parse()
- Qore::Program::parseCommit()
- Qore::Program::parseRollback()
- Qore::parse()

Since

Qore 0.8.7 the *source*, *offset*, and *format_label* arguments were added

45.41.2.27 nothing Qore::Program::parseRollback ( )

Rolls back any pending code processed with Program::parsePending() that has not yet been committed to the Program object with Program::parseCommit()

Example:

```
$pgm.parseRollback();
```

See also

- Qore::Program::parse()
- Qore::Program::parseCommit()
- Qore::Program::parsePending()
- Qore::parse()

45.41.2.28 nothing Qore::Program::replaceParseOptions (softint *opt*)

Replaces the parse options for the Program object.

An `OPTION-ERROR` exception is thrown if the calling Program object does not have PO_NO_CHILD_PO_RESTRICTIONS set.

Parameters

opt	A single parse option or binary-or'ed combination of parse options to unset in the parse option mask for the object

Example:

```
# disallow threading and GUI operations
$pgm.replaceParseOptions(PO_NO_THREADS | PO_NO_GUI);
```

Exceptions

OPTION-ERROR	The calling Program does not have the PO_NO_CHILD_PO_RESTRICTIONS option set, and therefore cannot call Program::replaceParseOptions()

See also

Program::setParseOptions() and Program::disableParseOptions().

45.41.2.29 any Qore::Program::run ( )

Runs the program and optionally returns a value if the top-level code exits with a return statement.

Returns

the value given to the return statement at the top-level, if any, otherwise NOTHING

Example:

```
$pgm.run();
```

45.41.2.30 nothing Qore::Program::setParseOptions (softint *opt* = **PO_DEFAULT**)

Sets parse options in the parse option mask for the Program object.

An `OPTIONS-LOCKED` exception is thrown if parse options have been locked (for example with Program::lockOptions())

Parameters

opt	A single parse option or binary-or'ed combination of parse options to set in the parse option mask for the object; the given argument will be combined with binary or with the existing parse option mask

Example:

```
# disable threading and GUI operations
$pgm.setParseOptions(PO_NO_THREADS | PO_NO_GUI);
```

Exceptions

OPTIONS-LOCKED	Parse options have been locked and cannot be changed
PROGRAM-OPTION-ERROR	invalid parse options used

See also

Program::disableParseOptions() for a reciprocal method that disables parse options; also see Program::replaceParseOptions()

45.41.2.31 nothing Qore::Program::setScriptPath (__7_ string *path*)

Sets (or clears) the script path (directory and filename) for the object.

Parameters

path	The path (directory and filename) for the current script; if the directory component is missing, then the current directory is assumed

Example:

```
$pgm.setScriptPath("/users/test/test.q");
```

45.41.2.32 nothing Qore::Program::setTimeZone (TimeZone *zone*)

Sets the default local time zone for the object.

Example:

```
my TimeZone $tz("Europe/Prague");
$pgm.setTimeZone($tz);
```

See also

TimeZone::set()

45.41.2.33 nothing Qore::Program::setTimeZoneRegion (string *region*)

Sets the default local time zone for the object from a path to a zoneinfo time zone region file.

If there are errors opening, reading, or parsing the file (or the Windows registry entry, depending on the platform), an exception is thrown

Parameters

region	The path to the zoneinfo file for the time zone region to set as the local time zone for the Program object

Example:

```
$pgm.setTimeZoneRegion("Europe/Prague");
```

Exceptions

TZINFO-ERROR	Unable to read zoneinfo file; invalid file magic; error parsing zoneinfo file, etc

45.41.2.34 nothing Qore::Program::setTimeZoneUTCOffset (softint *seconds_east*)

Sets the default time zone for the Program object based on the number of seconds east of UTC; for zones west of UTC, use negative numbers.

Time zones set with this method cannot have any daylight savings time information; to set a zone with daylight savings time information, use Program::setTimeZoneRegion() instead

Parameters

seconds_east	The number of seconds east of UTC; for zones west of UTC, use negative numbers

Example:

The following examples are all equivalent, setting the time zone to +02 UTC:

```
$pgm.setTimeZoneUTCOffset(7200);

$pgm.setTimeZoneUTCOffset(2h);

$pgm.setTimeZoneUTCOffset(PT2H);
```

See also

TimeZone::setUTCOffset()

45.41.2.35 nothing Qore::Program::undefine (string *def*)

Unsets a parse define for the current Program.

Parameters

def	The name of the define to undefine; if the given define is not defined anyway, the operation is ignored

Example:

```
$pgm.undefine("PRODUCTION");
```

45.42 Qore::Thread::Queue Class Reference

Queue objects provide a blocking, thread-safe message-passing object to Qore programs

Public Member Functions

- nothing clear ()

  Clears the Queue of all data.
- constructor (int max=-1)

  Creates the Queue object.
- copy ()

  Creates a new Queue object with the same elements and maximum size as the original.
- destructor ()

Destroys the Queue object.

- bool empty ()

  Returns True if the Queue is empty, False if not.

- any get (timeout timeout_ms=0)

  Blocks until at least one entry is available on the queue, then returns the first entry in the queue. If a timeout occurs, an exception is thrown. If the timeout is less than or equal to zero, then the call does not timeout until data is available.

- int getReadWaiting ()

  Returns the number of threads currently blocked on this queue for reading.

- int getWaiting ()

  Returns the number of threads currently blocked on this queue for reading.

- int getWriteWaiting ()

  Returns the number of threads currently blocked on this queue for writing.

- nothing insert (any arg, timeout timeout_ms=0)

  Inserts a value at the beginning of the queue.

- int max ()

  Returns the upper limit of the number of elements in the Queue.

- any pop (timeout timeout_ms=0)

  Blocks until at least one entry is available on the queue, then returns the last entry in the queue. If a timeout occurs, an exception is thrown. If the timeout is less than or equal to zero, then the call does not timeout until data is available.

- nothing push (any arg, timeout timeout_ms=0)

  Pushes a value on the end of the queue.

- int size ()

  Returns the number of elements in the Queue.

45.42.1 Detailed Description

Queue objects provide a blocking, thread-safe message-passing object to Qore programs

Restrictions:

Qore::PO_NO_THREAD_CLASSES

Queue objects can also be used as a stack or as a blocking message channel, if a maximum size is given to Queue::constructor() when the object is created. In this case when the Queue is full, adding new elements to the Queue will block until the Queue shrinks below the maximum size. All read and write methods to Queue also take timeout values; if a timeout occurs a `QUEUE-TIMEOUT` exception is thrown.

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option

45.42.2 Member Function Documentation

45.42.2.1 nothing Qore::Thread::Queue::clear ( )

Clears the Queue of all data.

Example:

```
$queue.clear();
```

Note

This method does not throw any exceptions, but exceptions could be thrown by in destructors of objects that go out of scope by being removed from the Queue

45.42.2.2 Qore::Thread::Queue::constructor (int *max* = -1)

Creates the Queue object.

Example:

```
my Queue $queue();
```

Parameters

max	the maximum size of the Queue; -1 means no limit; if 0 or a negative number other than -1 is passed then a QUEUE-SIZE-ERROR exception will be thrown

Exceptions

QUEUE-SIZE-ERROR	the size cannot be zero or any negative number except for -1 or a number that cannot fit in 32 bits (signed)

See also

Queue::max()

Since

Qore 0.8.4 this method takes a maximum size parameter and can throw exceptions if the parameter is invalid

45.42.2.3 Qore::Thread::Queue::destructor ( )

Destroys the Queue object.

Note

It is a programming error to delete this object while other threads are blocked on it; in this case an exception is thrown in the deleting thread, and also in each thread blocked on this object when it is deleted

Exceptions

QUEUE-ERROR	The queue was deleted while at least one thread was blocked on it

45.42.2.4 bool Qore::Thread::Queue::empty ( )

Returns True if the Queue is empty, False if not.

Code Flags:

CONSTANT

Example:

```
my bool $b = $queue.empty();
```

Returns

True if the Queue is empty, False if not

See also

Queue::size()

Since

Qore 0.8.8

45.42.2.5 any Qore::Thread::Queue::get (timeout *timeout_ms* = 0)

Blocks until at least one entry is available on the queue, then returns the first entry in the queue. If a timeout occurs, an exception is thrown. If the timeout is less than or equal to zero, then the call does not timeout until data is available.

Example:

```
my any $data = $queue.get();
```

Parameters

timeout_ms	a timeout value to wait for data to become available on the queue; integers are interpreted as milliseconds; relative date/time values are interpreted literally with a maximum resolution of milliseconds. Values <= 0 mean do not timeout. If a non-zero timeout argument is passed, and no data is available in the timeout period, a "QUEUE-TIMEOUT" exception is thrown. If no value or a value that converts to integer 0 is passed as the argument, then the call does not timeout until data is available on the queue.

Returns

the first entry on the queue

Note

This method throws a "QUEUE-TIMEOUT" exception on timeout, in order to enable the case where NOTHING was pushed on the queue to be differentiated from a timeout

Exceptions

QUEUE-TIMEOUT	The timeout value was exceeded
QUEUE-ERROR	The queue was deleted while at least one thread was blocked on it

45.42.2.6 int Qore::Thread::Queue::getReadWaiting ( )

Returns the number of threads currently blocked on this queue for reading.

This is a "synonym" for Queue::getWaiting()

Code Flags:

CONSTANT

Example:

```
my int $waiting = $queue.numReadWaiting();
```

Returns

the number of threads currently blocked on this queue for reading

See also

Queue::getWriteWaiting()

Since

Qore 0.8.4

45.42.2.7 int Qore::Thread::Queue::getWaiting ( )

Returns the number of threads currently blocked on this queue for reading.

This is a "synonym" for Queue::getReadWaiting()

Code Flags:

CONSTANT

Example:

```
my int $waiting = $queue.numWaiting();
```

Returns

the number of threads currently blocked on this queue for reading

See also

Queue::getWriteWaiting()

45.42.2.8 int Qore::Thread::Queue::getWriteWaiting ( )

Returns the number of threads currently blocked on this queue for writing.

Code Flags:

CONSTANT

Example:

```
my int $waiting = $queue.getWriteWaiting();
```

Returns

the number of threads currently blocked on this queue for writing

See also

Queue::getReadWaiting()

Since

Qore 0.8.4

45.42.2.9 nothing Qore::Thread::Queue::insert (any *arg*, timeout *timeout_ms* = 0)

Inserts a value at the beginning of the queue.

Example:

```
$queue.insert($value);
```

Parameters

arg	value to be put on the queue
timeout_ms	a timeout value to wait for a free entry to become available on the queue; integers are interpreted as milliseconds; relative date/time values are interpreted literally with a maximum resolution of milliseconds. Values <= 0 mean do not timeout. If a non-zero timeout argument is passed, and no data is available in the timeout period, a "QUEUE-TIMEOUT" exception is thrown. If no value or a value that converts to integer 0 is passed as the argument, then the call does not timeout until a slot becomes available on the queue. Queue slots are only limited if a maximum size is passed to Queue::constructor().

Exceptions

QUEUE-TIMEOUT	The timeout value was exceeded
QUEUE-ERROR	The queue was deleted while at least one thread was blocked on it

Since

Qore 0.8.4 this method takes a timeout parameter

45.42.2.10 int Qore::Thread::Queue::max ( )

Returns the upper limit of the number of elements in the Queue.

Code Flags:

CONSTANT

Example:

```
my int $max = $queue.max();
```

Returns

the upper limit of the number of elements in the Queu

See also

Queue::size()

45.42.2.11 any Qore::Thread::Queue::pop (timeout *timeout_ms* = 0)

Blocks until at least one entry is available on the queue, then returns the last entry in the queue. If a timeout occurs, an exception is thrown. If the timeout is less than or equal to zero, then the call does not timeout until data is available.

Example:

```
my any $data = $queue.pop();
```

Parameters

timeout_ms	a timeout value to wait for data to become available on the queue; integers are interpreted as milliseconds; relative date/time values are interpreted literally with a maximum resolution of milliseconds. Values <= 0 mean do not timeout. If a non-zero timeout argument is passed, and no data is available in the timeout period, a "QUEUE-TIMEOUT" exception is thrown. If no value or a value that converts to integer 0 is passed as the argument, then the call does not timeout until data is available on the queue.

Returns

the last entry on the queue

Note

This method throws a "QUEUE-TIMEOUT" exception on timeout, in order to enable the case where NOTHING was pushed on the queue to be differentiated from a timeout

Exceptions

QUEUE-TIMEOUT	The timeout value was exceeded
QUEUE-ERROR	The queue was deleted while at least one thread was blocked on it

45.42.2.12 nothing Qore::Thread::Queue::push (any *arg*, timeout *timeout_ms* = 0)

Pushes a value on the end of the queue.

Example:

```
$queue.push($value);
```

Parameters

arg	value to be put on the queue
timeout_ms	a timeout value to wait for a free entry to become available on the queue; integers are interpreted as milliseconds; relative date/time values are interpreted literally with a maximum resolution of milliseconds. Values <= 0 mean do not timeout. If a non-zero timeout argument is passed, and no data is available in the timeout period, a "QUEUE-TIMEOUT" exception is thrown. If no value or a value that converts to integer 0 is passed as the argument, then the call does not timeout until a slot becomes available on the queue. Queue slots are only limited if a maximum size is passed to Queue::constructor().

Exceptions

QUEUE-TIMEOUT	The timeout value was exceeded
QUEUE-ERROR	The queue was deleted while at least one thread was blocked on it

Since

Qore 0.8.4 this method takes a timeout parameter

45.42.2.13 int Qore::Thread::Queue::size ( )

Returns the number of elements in the Queue.

Code Flags:

CONSTANT

Example:

```
my int $size = $queue.size();
```

Returns

the number of elements in the Queue

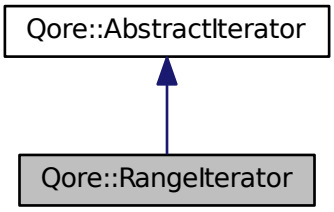
See also

Queue::max()

45.43 Qore::RangeIterator Class Reference

This class defines a range-like iterator to be used to iterate numerical sequences.

Inheritance diagram for Qore::RangeIterator:



Public Member Functions

- constructor (int start, int stop, int step=1)

  creates the numerical sequence iterator with the initial arguments
- copy ()

  Creates a copy of the RangeIterator object, iterating the same object as the original and in the same position.
- any getValue ()

  returns the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- bool next ()

  This method returns True while there are more numbers to iterate and False when the range has been completely iterated.
- reset ()

  Reset the iterator instance to its initial state (start, stop, and step).
- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.43.1 Detailed Description

This class defines a range-like iterator to be used to iterate numerical sequences.

The main purpose is to provide resource friendly iterator to generate numerical rows (sequences) with ascending and descending ordering.

The RangeIterator class provides an iterator for loop statements with functionality similar to range(). Unlike range() RangeIterator objects do not generate real lists but calculate iteration values on demand.

This results in memory-friendly handling for large numerical sequences compared to generating a list in memory and iterating that list (as with Qore::range()).

Example: RangeIterator basic usage

```
my RangeIterator $r(2);
for my int $i in ($r)
    printf("i=%d\n", $i);
# i=0
# i=1
# i=2
```

See also

range()
xrange()

Since

Qore 0.8.6

45.43.2 Member Function Documentation

45.43.2.1 Qore::RangeIterator::constructor (int *start*, int *stop*, int *step* = 1)

creates the numerical sequence iterator with the initial arguments

Parameters

start	an initial value
stop	a final value
step	is the interval. Default = 1

Example:

```
my RangeIterator $i(5, 10, 2);
```

See also

xrange()

45.43.2.2 Qore::RangeIterator::copy ( )

Creates a copy of the RangeIterator object, iterating the same object as the original and in the same position.

Example:

```
my RangeIterator $ni = $i.copy();
```

45.43.2.3 any Qore::RangeIterator::getValue ( ) [virtual]

returns the current value or throws an INVALID-ITERATOR exception if the iterator is invalid

Returns

the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element
ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

45.43.2.4 bool Qore::RangeIterator::next ( ) [virtual]

This method returns True while there are more numbers to iterate and False when the range has been completely iterated.

The iterator object should not be used after this method returns False.

Returns

True and False alternately unless it has no value iterate.

Example:

```
while ($i.next()) {
    printf("value: %y\n", $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

45.43.2.5 Qore::RangeIterator::reset ( )

Reset the iterator instance to its initial state (start, stop, and step).

Reset the iterator instance to its initial state (start, stop, and step).

Example

```
# raw RangeIterator object usage
my RangeIterator $r(2);
foreach my int $i in ($r) {
    printf("i=%d\n", $i);
    if ($i == 1)
        break;
}
# i=0
# i=1

# show the reset feature; start iterating all over again
$r.reset();
foreach my int $i in ($r)
```

```
        printf("reused i=%d\n", $i);
    # reused i=0
    # reused i=1
    # reused i=2
```

Exceptions

ITERATOR-THREAD-ERROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.43.2.6 bool Qore::RangeIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

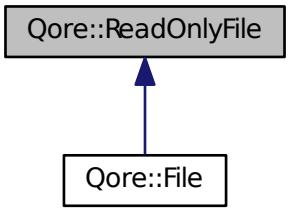
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.44 Qore::ReadOnlyFile Class Reference

The ReadOnlyFile class allows Qore programs to read existing files.

Inheritance diagram for Qore::ReadOnlyFile:



Public Member Functions

- int close ()

  Closes the ReadOnlyFile object.
- constructor (string path, __7_ string encoding)

  Creates the ReadOnlyFile object.
- copy ()

  Creates a new ReadOnlyFile object with the same character encoding specification as the original, otherwise no other information is copied.
- destructor ()

  Closes the ReadOnlyFile if it is open and destroys the ReadOnlyFile object.

- string getEncoding ()

  Returns the character encoding for the ReadOnlyFile.
- __7_ string getFileName ()

  returns the file path/name used to open the file if the file is open, otherwise NOTHING
- int getPos ()

  Returns the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)
- __7_ string getchar ()

  Reads one character from the file and returns it as a string; returns NOTHING if no data can be read from the file.
- hash hstat ()

  Returns a Stat Hash about the file's status or throws an exception if any errors occur.
- bool isDataAvailable (timeout timeout_ms=0)

  Returns True if there is data available for reading from the file within the timeout period.
- bool isOpen ()

  returns True if the File is open, False if not
- bool isTty ()

  returns True if the File is connected to a terminal device, False if not
- nothing open (string path, __7_ string encoding)

  Opens a file in a particular mode; throws an exception on failure.
- __7_ string read (softint size, timeout timeout_ms=-1)

  Reads a certain number of bytes from the ReadOnlyFile within an optional timeout period and returns a string of the data read or NOTHING if no data can be read.
- __7_ binary readBinary (softint size, timeout timeout_ms=-1)

  Reads a certain number of bytes from the file within an optional timeout period and returns a binary object of the data read or NOTHING if no data can be read.
- __7_ string readLine (bool incl_eol=True, __7_ string eol)

  Reads until an EOL marker is found and returns the string read or NOTHING if no data can be read.
- __7_ int readi1 ()

  Reads a 1-byte signed integer from the file in binary format or NOTHING if no data can be read.
- __7_ int readi2 ()

  Reads a 2-byte (16 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.
- __7_ int readi2LSB ()

  Reads a 2-byte (16 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.
- __7_ int readi4 ()

  Reads a 4-byte (32 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.
- __7_ int readi4LSB ()

  Reads a 4-byte (32 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.
- __7_ int readi8 ()

  Reads an 8-byte (64 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.
- __7_ int readi8LSB ()

  Reads an 8-byte (64 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.
- __7_ int readu1 ()

  Reads a 1-byte unsigned integer from the ReadOnlyFile in binary format or NOTHING if no data can be read.
- __7_ int readu2 ()

  Reads a 2-byte (16 bit) unsigned integer from the ReadOnlyFile in binary big-endian format or NOTHING if no data can be read.
- __7_ int readu2LSB ()

  Reads a 2-byte (16 bit) unsigned integer from the file in binary little-endian format or NOTHING if no data can be read.
- __7_ int readu4 ()

  Reads a 4-byte (32 bit) unsigned integer from the file in big-endian format or NOTHING if no data can be read.
- __7_ int readu4LSB ()

Reads a 4-byte (32 bit) unsigned integer from the file in binary little-endian format or NOTHING if no data can be read.

- nothing setEncoding (__7_ string encoding)

  Sets the character encoding for the ReadOnlyFile; if called with no argument, the default encoding is set.
- nothing setEventQueue (Qore::Thread::Queue queue)

  Sets a Queue object to receive file events.
- nothing setEventQueue ()

  Removes any Queue object from the ReadOnlyFile object so that file events are no longer added to the Queue.
- int setPos (int pos=0)

  Sets the current file position (in bytes from the beginning of the file)
- list stat ()

  Returns a Stat List about the file's status or throws an exception if any errors occur.
- hash statvfs ()

  Returns a Filesystem Status Hash about the file's filesystem status or throws an exception if any errors occur.

Static Public Member Functions

- static binary readBinaryFile (string path)

  returns the contents of a binary file as a binary object
- static string readTextFile (string path, __7_ string encoding)

  returns the contents of a text file as a string optionally tagged with the given character encoding

45.44.1 Detailed Description

The ReadOnlyFile class allows Qore programs to read existing files.

Restrictions:

Qore::PO_NO_FILESYSTEM

Note

This class is not available with the PO_NO_FILESYSTEM parse option

ReadOnlyFile objects are opened with a specific character encoding, meaning that any string read from the file will be tagged with the file's character encoding. If no character encoding is specified, then the default character encoding is assumed for the file.

This class supports posting read events to a Queue. See I/O Event Handling for more information.

See also

file_events for a list of I/O events raised by this object

Since

Qore 0.8.6 this class was split from the Qore::File class and added as a base class of Qore::File

45.44.2 Member Function Documentation

45.44.2.1 int Qore::ReadOnlyFile::close ( )

Closes the ReadOnlyFile object.

Example:

```
if ($f.close())
    printf("error closing file: %s\n", strerror(errno()));
```

Events:

EVENT_CHANNEL_CLOSED

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

Returns

0 for success, -1 for an error (see errno() and strerror() for the error information)

45.44.2.2 Qore::ReadOnlyFile::constructor (**string** *path*, __7_ **string** *encoding*)

Creates the ReadOnlyFile object.

The constructor takes the file path and one optional argument that will set the default character encoding for the file (only affects reading string data) To open the file again with another file, call ReadOnlyFile::open(); the character encoding can also be set or changed by the ReadOnlyFile::open() or ReadOnlyFile::setEncoding() methods.

Example:

```
my ReadOnlyFile $f("/tmp/my-file.txt", "iso-8859-1");
```

Parameters

path	the path to open for reading
encoding	The character encoding for the ReadOnlyFile. Strings data read from the file will be tagged with this character encoding; if this argument is not given then the ReadOnlyFile will receive the default encoding

Exceptions

READONLYFILE-OPEN-ERROR	the given file cannot be opened for reading (`arg` will be assigned to the errno value)
ILLEGAL-EXPRESSION	ReadOnlyFile::constructor() cannot be called with a TTY target when %no-terminal-io is set

See also

ReadOnlyFile::open()

45.44.2.3 Qore::ReadOnlyFile::copy ( )

Creates a new ReadOnlyFile object with the same character encoding specification as the original, otherwise no other information is copied.

Example:

```
my ReadOnlyFile $f1 = $f.copy();
```

45.44.2.4 Qore::ReadOnlyFile::destructor ( )

Closes the ReadOnlyFile if it is open and destroys the ReadOnlyFile object.

Closes the ReadOnlyFile if it is open and destroys the ReadOnlyFile object

45.44.2.5 __7_ string Qore::ReadOnlyFile::getchar ( )

Reads one character from the file and returns it as a string; returns NOTHING if no data can be read from the file.

Multi-byte characters are also read; use ReadOnlyFile::readu1() or ReadOnlyFile::readi1() to read a single byte from a file regardless of the ReadOnlyFile's character encoding.

Example:

```
my *string $str = $f.getchar();
```

Returns

the single character read from the file or NOTHING if no data can be read from the ReadOnlyFile

Exceptions

FILE-READ-ERROR	file is not open
FILE-GETCHAR-ERROR	invalid multi-byte character read or EOF received while reading a multi-byte character
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.6 string Qore::ReadOnlyFile::getEncoding ( )

Returns the character encoding for the ReadOnlyFile.

Code Flags:

CONSTANT

Example:

```
my string $encoding = $f.getEncoding();
```

Returns

the character encoding for the ReadOnlyFile

45.44.2.7 __7_ string Qore::ReadOnlyFile::getFileName ( )

returns the file path/name used to open the file if the file is open, otherwise NOTHING

Code Flags:

CONSTANT

Example:

```
my *string $fn = $f.getFileName();
```

Returns

the file path/name used to open the file if the file is open, otherwise NOTHING

Since

Qore 0.8.6

45.44.2.8 int Qore::ReadOnlyFile::getPos ( )

Returns the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $pos = $f.getPos();
```

Returns

the current file position as an integer giving the offset in bytes from the beginning of the file (starting from zero)

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

ReadOnlyFile::setPos()

45.44.2.9 hash Qore::ReadOnlyFile::hstat ( )

Returns a Stat Hash about the file's status or throws an exception if any errors occur.

If any errors occur, a `FILE-HSTAT-ERROR` exception is thrown

Example:

```
my hash $h = $file.hstat();
```

Returns

a Stat Hash about the file's status

Exceptions

FILE-HSTAT-ERROR	stat() call failed or file not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

- File::stat()
- File Stat Constants

45.44.2.10 bool Qore::ReadOnlyFile::isDataAvailable (timeout *timeout_ms* = 0)

Returns True if there is data available for reading from the file within the timeout period.

With a timeout of zero (the default if no timeout value is passed), this method can be used for non-blocking polling the file for data. Like all Qore functions and methods taking timeout values, a relative date/time value may be passed instead of an integer to make the timeout units clear (ex: `25ms`).

Code Flags:

RET_VALUE_ONLY

Example:

```
if (!isDataAvailable(30s))
    return;
```

Parameters

timeout_ms	An optional timeout in milliseconds (1/1000 second); relative date/time values can be given instead of an integer in milliseconds to make the source more readable; ex: `20s`

Returns

True if data becomes available for reading from the file within the timeout period, False if not

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.11 bool Qore::ReadOnlyFile::isOpen ( )

returns True if the File is open, False if not

Returns

True if the File is open, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $file.isOpen();
```

45.44.2.12 bool Qore::ReadOnlyFile::isTty ( )

returns True if the File is connected to a terminal device, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $file.isTty();
```

Returns

True if the File is connected to a terminal device, False if not

Since

Qore 0.8.6

45.44.2.13 nothing Qore::ReadOnlyFile::open (string *path*, __7_ string *encoding*)

Opens a file in a particular mode; throws an exception on failure.

Opens the file in the mode given; if the ReadOnlyFile was previously open, it is closed first. Optionally the ReadOnlyFile's default character encoding can be specified.

Note that if no encoding is specified, the ReadOnlyFile will be tagged with the character encoding set in the ReadOnlyFile's constructor. Any string data written to the ReadOnlyFile will be converted to the ReadOnlyFile's encoding, and any string data read from the ReadOnlyFile will be automatically tagged with the ReadOnlyFile's encoding.

If an error occurs, a `READONLYFILE-OPEN-ERROR` exception is thrown.

Example:

```
try {
    $f.open($fn, "iso-8859-1");
}
catch (hash $ex) {
    printf("%s: %s: %s\n", $fn, $ex.err, $ex.desc);
}
```

Events:

EVENT_OPEN_FILE, EVENT_FILE_OPENED

Parameters

path	the path to the file
encoding	the name of the character encoding for the ReadOnlyFile; if this argument is not given, the ReadOnlyFile will be tagged with the character encoding given in the constructor

Exceptions

READONLYFILE-OPEN-ERROR	an error occurred opening the file
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.14 __7_ string Qore::ReadOnlyFile::read (softint *size*, timeout *timeout_ms* = `-1`)

Reads a certain number of bytes from the ReadOnlyFile within an optional timeout period and returns a string of the data read or NOTHING if no data can be read.

Reads a certain amount of string data from the ReadOnlyFile; the size argument is required. To read binary data, use the ReadOnlyFile::readBinary() method.

Note that the amount of data read from the file may be less than the size given, for example if the file does not contain enough data to fulfill the request. In this case, only the data available in the file is returned.

An optional timeout period in milliseconds can be passed as well (or a relative date/time value may be passed instead of an integer to make the timeout units clear; ex: `25ms`). If a timeout value is passed and the data cannot be read within the timeout period, then a `READONLYFILE-READ-TIMEOUT` exception is thrown. If no timeout value is passed or a negative value is given, then the call will never timeout until either the requested amount of data has been read from the ReadOnlyFile or an end-of-file condition has been reached.

Example:

```
my *string $data = $f.read(-1); # read an entire text file into a variable
```

Events:

EVENT_DATA_READ

Parameters

size	the number of bytes to read of the file, -1 will read the entire file
timeout_ms	a timeout period with a resolution of milliseconds (a relative date/time value; integer arguments will be assumed to be milliseconds); if not given or negative the call will never time out and will only return when the data has been read

Returns

the data read from the file, returned as a string tagged with the ReadOnlyFile's character encoding. NOTHING is returned if end-of-file is encountered, however, if data has been read before EOF, the data read will be returned and NOTHING (signifying EOF) will be returned on the next call to this method.

Exceptions

READONLYFILE-READ-PARAMETER-ERROR	zero size argument passed
FILE-READ-ERROR	file is not open
FILE-READ-TIMEOUT	timeout limit exceeded
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

File::readBinary()

45.44.2.15 __7_ binary Qore::ReadOnlyFile::readBinary (softint *size*, timeout *timeout_ms* = -1)

Reads a certain number of bytes from the file within an optional timeout period and returns a binary object of the data read or NOTHING if no data can be read.

Reads a certain amount of string data from the file; the size argument is required. To read string data, use the ReadOnlyFile::read() method.

Note that the amount of data read from the file may be less than the size given, for example if the file does not contain enough data to fulfill the request. In this case, only the data available in the file is returned.

An optional timeout period in milliseconds can be passed as well (or a relative date/time value may be passed instead of an integer to make the timeout units clear; ex: `25ms`). If a timeout value is passed and the data cannot be read within the timeout period, then a `FILE-READ-TIMEOUT` exception is thrown. If no timeout value is passed or a negative value is given, then the call will never timeout until either the requested amount of data has been read from the file or an end-of-file condition has been reached.

Events:

EVENT_DATA_READ

Example:

```
my *binary $data = $f.readBinary(-1); # read an entire file into a variable
```

Parameters

size	the number of bytes to read of the file, -1 will read the entire file
timeout_ms	a timeout period with a resolution of milliseconds (a relative date/time value; integer arguments will be assumed to be milliseconds); if not given or negative the call will never time out and will only return when the data has been read

Returns

the data read from the file, returned as a binary object. NOTHING is returned if end-of-file is encountered, however, if data has been read before EOF, the data read will be returned and NOTHING (signifying EOF) will be returned on the next call to this method.

Exceptions

READONLYFILE-READ-BINARY-PARAMETER-ERROR	zero size argument passed
FILE-READ-ERROR	file is not open
FILE-READ-TIMEOUT	timeout limit exceeded
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

ReadOnlyFile::read()

45.44.2.16 static binary Qore::ReadOnlyFile::readBinaryFile (string *path*) [static]

returns the contents of a binary file as a binary object

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my binary $data = File::readBinaryFile($path);
```

Parameters

path	the path of the file to retrieve

Returns

the contents of a binary file as a binary object

Since

Qore 0.8.8

45.44.2.17 __7_ int Qore::ReadOnlyFile::readi1 ( )

Reads a 1-byte signed integer from the file in binary format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi1();
```

Events:

EVENT_DATA_READ

Returns

a 1-byte signed integer as read from the file in binary format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.18 __7_ int Qore::ReadOnlyFile::readi2 ( )

Reads a 2-byte (16 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi2();
```

Events:

EVENT_DATA_READ

Returns

a 2-byte signed integer as read from the file in binary big-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.19 __7_ int Qore::ReadOnlyFile::readi2LSB ( )

Reads a 2-byte (16 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi2LSB();
```

Events:

EVENT_DATA_READ

Returns

a 2-byte signed integer as read from the file in binary little-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.20 __7_ int Qore::ReadOnlyFile::readi4 ( )

Reads a 4-byte (32 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi4();
```

Events:

EVENT_DATA_READ

Returns

a 4-byte signed integer as read from the file in binary big-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.21 __7_ int Qore::ReadOnlyFile::readi4LSB ( )

Reads a 4-byte (32 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi4LSB();
```

Events:

EVENT_DATA_READ

Returns

a 4-byte signed integer as read from the file in binary little-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.22 __7_ int Qore::ReadOnlyFile::readi8 ( )

Reads an 8-byte (64 bit) signed integer from the file in binary big-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi8();
```

Events:

EVENT_DATA_READ

Returns

a 8-byte signed integer as read from the file in binary big-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.23 __7_ int Qore::ReadOnlyFile::readi8LSB ( )

Reads an 8-byte (64 bit) signed integer from the file in binary little-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readi8LSB();
```

Events:

EVENT_DATA_READ

Returns

an 8-byte signed integer as read from the file in binary little-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.24 __7_ string Qore::ReadOnlyFile::readLine (bool *incl_eol* = True, __7_ string *eol*)

Reads until an EOL marker is found and returns the string read or NOTHING if no data can be read.

Any string returned will be tagged with the ReadOnlyFile's character encoding.

Example:

```
# remove EOL marker
while (exists (my *string $line = $f.readLine(False))) {
    # print out the line just read
    printf("%s\n", $line);
}
```

Events:

EVENT_DATA_READ

Parameters

incl_eol	if this argument is True, then the end of line characters read are included in the string returned
eol	the end of line character(s); if not given, then the end of line character(s) are detected automatically, and can be either `"\n"`, `"\r"`, or `"\r\n"` (the last one is only automatically detected when not connected to a terminal device in order to keep the I/O from stalling); if this string is passed and has a different character encoding from the File's, then it will be converted to the File's character encoding

Returns

The line read from the file. NOTHING is returned if end-of-file is encountered, however, if data has been read before EOF, the data read will be returned and NOTHING (signifying EOF) will be returned on the next call to this method. Any string returned will be tagged with the ReadOnlyFile's character encoding.

Exceptions

READONLYFILE-READLINE-ERROR	the file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

- Qore::FileLineIterator

45.44.2.25 static string Qore::ReadOnlyFile::readTextFile (string *path*, __7_ string *encoding*) [static]

returns the contents of a text file as a string optionally tagged with the given character encoding

Restrictions:

Qore::PO_NO_FILESYSTEM

Example:

```
my string $data = File::readTextFile($path);
```

Parameters

path	the path of the file to retrieve
encoding	the name of the character encoding for the string returned; if this argument is not given, then the default character encoding is assumed

Returns

the contents of a text file as a string optionally tagged with the given character encoding

Since

Qore 0.8.8

45.44.2.26 __7_ int Qore::ReadOnlyFile::readu1 ( )

Reads a 1-byte unsigned integer from the ReadOnlyFile in binary format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readu1();
```

Events:

EVENT_DATA_READ

Returns

a 1-byte unsigned integer as read from the ReadOnlyFile in binary format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.27 __7_ int Qore::ReadOnlyFile::readu2 ( )

Reads a 2-byte (16 bit) unsigned integer from the ReadOnlyFile in binary big-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readu2();
```

Events:

EVENT_DATA_READ

Returns

a 2-byte unsigned integer as read from the file in binary big-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.28 __7_ int Qore::ReadOnlyFile::readu2LSB ( )

Reads a 2-byte (16 bit) unsigned integer from the file in binary little-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readu2LSB();
```

Events:

EVENT_DATA_READ

Returns

a 2-byte unsigned integer as read from the file in binary little-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.29 __7_ int Qore::ReadOnlyFile::readu4 ( )

Reads a 4-byte (32 bit) unsigned integer from the file in big-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readu4();
```

Events:

EVENT_DATA_READ

Returns

a 4-byte unsigned integer as read from the file in binary big-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.30 __7_ int Qore::ReadOnlyFile::readu4LSB ( )

Reads a 4-byte (32 bit) unsigned integer from the file in binary little-endian format or NOTHING if no data can be read.

Example:

```
my *int $i = $f.readu4LSB();
```

Events:

EVENT_DATA_READ

Returns

a 4-byte unsigned integer as read from the file in binary little-endian format or NOTHING if no data can be read

Exceptions

FILE-READ-ERROR	file is not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.31 nothing Qore::ReadOnlyFile::setEncoding (__7_ string *encoding*)

Sets the character encoding for the ReadOnlyFile; if called with no argument, the default encoding is set.

Example:

```
$f.setEncoding("ISO-8859-1");
```

Parameters

encoding	the character encoding for the file; if called with no argument, the default encoding is set

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.44.2.32 nothing Qore::ReadOnlyFile::setEventQueue (Qore::Thread::Queue *queue*)

Sets a Queue object to receive file events.

Example:

```
$f.setEventQueue($queue);
```

Parameters

queue	the Queue object to receive file events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set
QUEUE-ERROR	the Queue passed has a maximum size set

See also

I/O Event Handling for more information

45.44.2.33 nothing Qore::ReadOnlyFile::setEventQueue ( )

Removes any Queue object from the ReadOnlyFile object so that file events are no longer added to the Queue.

Example:

```
$f.setEventQueue();
```

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

I/O Event Handling for more information

45.44.2.34 int Qore::ReadOnlyFile::setPos (int *pos* = 0)

Sets the current file position (in bytes from the beginning of the file)

Example:

```
$f.setPos(0); # go to the beginning of the file
```

Parameters

pos	the position in the file as offset from position 0

Returns

the new offset in the file, -1 for error

Exceptions

ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

ReadOnlyFile::getPos()

45.44.2.35 list Qore::ReadOnlyFile::stat ( )

Returns a Stat List about the file's status or throws an exception if any errors occur.

If any errors occur, a `FILE-STAT-ERROR` exception is thrown.

Example:

```
my int $mode = $file.stat()[2];
```

Returns

a list of file status values for the current file (must be open).

Exceptions

FILE-STAT-ERROR	stat() call failed or file not open
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

See also

- File::hstat()
- File Stat Constants

45.44.2.36 hash Qore::ReadOnlyFile::statvfs ( )

Returns a Filesystem Status Hash about the file's filesystem status or throws an exception if any errors occur.

If any errors occur, a `FILE-STATVFS-ERROR` exception is thrown

Platform Availability:

Qore::Option::HAVE_STATVFS

Example:

```
my hash $h = $file.statvfs();
```

Returns

a Filesystem Status Hash about the filesystem where the file resides

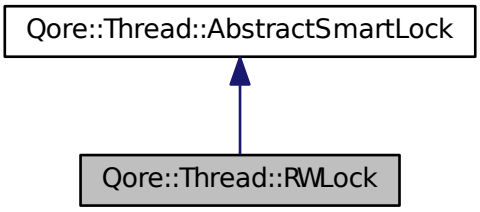
Exceptions

FILE-STATVFS-ERROR	statvfs() call failed or file is not open
MISSING-FEATURE-ERROR	this method is not supported on this platform; check Option::HAVE_STATVFS before calling this method to avoid this exception
ILLEGAL-EXPRESSION	this exception is only thrown if called with a system constant object (stdin, stdout, stderr) when no-terminal-io is set

45.45 Qore::Thread::RWLock Class Reference

The RWLock class implements a read-write thread lock.

Inheritance diagram for Qore::Thread::RWLock:



Public Member Functions

- constructor ()

  Creates the RWLock object.
- copy ()

  Creates a new RWLock object, not based on the original.
- destructor ()

  Destroys the RWLock object.
- int getReadWaiting ()

  Returns the number of threads waiting on the read lock.
- int getWriteWaiting ()

  Returns the number of threads waiting on the write lock.
- bool lockOwner ()

  Returns True if the current thread is holding either the read lock or the write lock, False if not.
- int numReaders ()

  Returns the read lock count.
- nothing readLock ()

  Acquires the read lock; blocks if the write lock is already acquired by another thread.
- int readLock (timeout timeout_ms)

  Acquires the read lock with a timeout value; blocks if the write lock is already acquired by another thread.
- bool readLockOwner ()

  Returns True if the current thread is holding the read lock, False if not.
- nothing readUnlock ()

Decrements the read lock counter and releases the read lock if the counter is zero. If at least one thread is blocked trying to acquire the write lock and the read counter reaches zero, then one thread waiting on the write lock is woken up.

- int tryReadLock ()

  Acquires the read lock only if it can be acquired immediately.
- int tryWriteLock ()

  Acquires the write lock only if it can be acquired immediately.
- int writeLock (timeout timeout_ms)

  Acquires the write lock with a timeout value; blocks if the read lock is already acquired by another thread.
- nothing writeLock ()

  Acquires the write lock; blocks if the read lock is already acquired by another thread.
- bool writeLockOwner ()

  Returns True if the current thread is holding the write lock, False if not.
- nothing writeUnlock ()

  Releases the write lock, if any readers are waiting, wakes up all readers, otherwise if any writers are waiting, then wakes one up.

45.45.1 Detailed Description

The RWLock class implements a read-write thread lock.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

This class inherits AbstractSmartLock, so it can be used by Condition objects, while using either the read lock or the write lock.

The RWLock class implements a read-write lock for efficient thread locking when write actions must be atomic and reads can be made in parallel if no write is in progress. When a thread holds the write lock, no other thread can grab the read or write lock. Multiple threads can hold the read lock at one time.

As with all Qore threading primitives, this class supports deadlock detection and throws exceptions when threading errors are encountered (for example, trying to free the read lock while holding the write lock, etc).

This read-write lock favors readers, so the read lock can be safely acquired recursively.

See the AutoReadLock and the AutoWriteLock classes for classes that assist in exception-safe RWLock locking.

Additionally, the on_exit statement can provide exception-safe RWLock handling at the lexical block level as in the following example:

```
{
    $rwl.writeLock();
    on_exit
        $rwl.writeUnlock();

    # ... when this block exits the lock will be released, even in the
    #     case of return statements or exceptions
}
```

Note

This class is not available with the PO_NO_THREAD_CLASSES parse option.

45.45.2 Member Function Documentation

45.45.2.1 Qore::Thread::RWLock::constructor ( )

Creates the RWLock object.

Example:

```
my RWLock $rwl();
```

45.45.2.2 Qore::Thread::RWLock::copy ( )

Creates a new RWLock object, not based on the original.

Example:

```
my RWLock $new_rwl = $rwl.copy();
```

45.45.2.3 Qore::Thread::RWLock::destructor ( )

Destroys the RWLock object.

Note that it is a programming error to delete this object while other threads are blocked on it; in this case an exception is thrown in the deleting thread, and in each thread blocked on this object when it is deleted.

Example:

```
delete $rwl;
```

Exceptions

LOCK-ERROR	Object deleted while other threads blocked on it

45.45.2.4 int Qore::Thread::RWLock::getReadWaiting ( )

Returns the number of threads waiting on the read lock.

Returns

the number of threads waiting on the read lock

Code Flags:

CONSTANT

Example:

```
my int $num = $rwl.getReadWaiting();
```

45.45.2.5 int Qore::Thread::RWLock::getWriteWaiting ( )

Returns the number of threads waiting on the write lock.

Returns

the number of threads waiting on the write lock

Code Flags:

CONSTANT

Example:

```
my int $num = $rwl.getWriteWaiting();
```

45.45.2.6 bool Qore::Thread::RWLock::lockOwner ( )

Returns True if the current thread is holding either the read lock or the write lock, False if not.

Returns

True if the current thread is holding either the read lock or the write lock, False if not

Code Flags:

CONSTANT

Example:

```
if ($rwl.lockOwner())
    printf("TID %d has either the read lock or the write lock\n", gettid());
```

45.45.2.7 int Qore::Thread::RWLock::numReaders ( )

Returns the read lock count.

Returns

the read lock count

Code Flags:

CONSTANT

Example:

```
my int $num = $rwl.numReaders();
```

45.45.2.8 nothing Qore::Thread::RWLock::readLock ( )

Acquires the read lock; blocks if the write lock is already acquired by another thread.

Example:

```
$rwl.readLock();
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::readLock() called while already holding the write lock, object deleted in another thread, etc.

45.45.2.9 int Qore::Thread::RWLock::readLock (timeout *timeout_ms*)

Acquires the read lock with a timeout value; blocks if the write lock is already acquired by another thread.

Returns 0 for success, non-zero for timeout; exceptions are thrown for other errors

Parameters

timeout_ms	a timeout value to wait to acquire the read lock; integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds)

Returns

0 for success, non-zero for timeout; exceptions are thrown for other errors

Example:

```
if ($rwl.readLock(1500ms))
    throw "TIMEOUT", "timed out after 1.5s waiting for the read lock";
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::readLock() called while already holding the write lock, object deleted in another thread, etc.

45.45.2.10 bool Qore::Thread::RWLock::readLockOwner ( )

Returns True if the current thread is holding the read lock, False if not.

Returns

True if the current thread is holding the read lock, False if not

Code Flags:

CONSTANT

Example:

```
if ($rwl.readLockOwner())
    printf("TID %d has the read lock\n", gettid());
```

45.45.2.11 nothing Qore::Thread::RWLock::readUnlock ( )

Decrements the read lock counter and releases the read lock if the counter is zero. If at least one thread is blocked trying to acquire the write lock and the read counter reaches zero, then one thread waiting on the write lock is woken up.

Example:

```
$rwl.readUnlock();
```

Exceptions

LOCK-ERROR	RWLock::readUnlock() called while not holding the read lock, object deleted in another thread, etc

45.45.2.12 int Qore::Thread::RWLock::tryReadLock ( )

Acquires the read lock only if it can be acquired immediately.

Returns

0 for success (read lock acquired, read lock count incremented) or -1 if the call would block (write lock owned by another thread) or an error occured

Example:

```
if (!$rwl.tryReadLock()) {
    on_exit $rwl.readUnlock();
    do_something_in_read_lock();
}
```

45.45.2.13 int Qore::Thread::RWLock::tryWriteLock ( )

Acquires the write lock only if it can be acquired immediately.

Returns

0 for success (write lock acquired) or -1 if the call would block (read lock owned by another thread) or an error occured

Example:

```
if (!$rwl.tryWriteLock()) {
    on_exit $rwl.writeUnlock();
    do_something_in_write_lock();
}
```

45.45.2.14 int Qore::Thread::RWLock::writeLock (timeout *timeout_ms*)

Acquires the write lock with a timeout value; blocks if the read lock is already acquired by another thread.

Returns 0 for success, non-zero for timeout; exceptions are thrown for other errors

Parameters

timeout_ms	a timeout value to wait to acquire the write lock; integers are interpreted as milliseconds; relative date/time values are interpreted literally (with a resolution of milliseconds)

Returns

0 for success, non-zero for timeout; exceptions are thrown for other errors

Example:

```
if ($rwl.writeLock(1500ms))
    throw "TIMEOUT", "timed out after 1.5s waiting for the write lock";
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::writeLock() called while already holding the read lock, object deleted in another thread, etc.

45.45.2.15 nothing Qore::Thread::RWLock::writeLock ( )

Acquires the write lock; blocks if the read lock is already acquired by another thread.

Example:

```
$rwl.writeLock();
```

Exceptions

THREAD-DEADLOCK	A deadlock was detected while trying to acquire the lock
LOCK-ERROR	RWLock::writeLock() called while already holding the read lock, object deleted in another thread, etc.

45.45.2.16 bool Qore::Thread::RWLock::writeLockOwner ( )

Returns True if the current thread is holding the write lock, False if not.

Returns

True if the current thread is holding the write lock, False if not

Code Flags:

CONSTANT

Example:

```
if ($rwl.writeLockOwner())
    printf("TID %d has the write lock\n", gettid());
```

45.45.2.17 nothing Qore::Thread::RWLock::writeUnlock ( )

Releases the write lock, if any readers are waiting, wakes up all readers, otherwise if any writers are waiting, then wakes one up.

Example:

```
$rwl.writeUnlock();
```

Exceptions

LOCK-ERROR	RWLock::writeUnlock() called while not holding the write lock, object deleted in another thread, etc

45.46 Qore::Thread::Sequence Class Reference

The Sequence class implements a thread-safe increment-only object.

Public Member Functions

- constructor ()

  Creates a new Sequence object.
- constructor (softint start)

  Creates a new Sequence object with a starting value.
- copy ()

  Creates a new Sequence object, not based on the original.
- int getCurrent ()

  Returns the current value of the sequence.
- int next ()

  Atomically increments the sequence value and returns the last value.

45.46.1 Detailed Description

The Sequence class implements a thread-safe increment-only object.

This class does not block therefore is not tagged with Qore::PO_NO_THREAD_CLASSES

45.46.2 Member Function Documentation

45.46.2.1 Qore::Thread::Sequence::constructor ( )

Creates a new Sequence object.

Example:

```
my Sequence $seq();
```

45.46.2.2 Qore::Thread::Sequence::constructor (softint *start*)

Creates a new Sequence object with a starting value.

Example:

```
my Sequence $seq(20);
```

45.46.2.3 Qore::Thread::Sequence::copy ( )

Creates a new Sequence object, not based on the original.

Example:

```
my Sequence $s2 = $seq.copy();
```

45.46.2.4 int Qore::Thread::Sequence::getCurrent ( )

Returns the current value of the sequence.

Returns

current value of the sequence

Code Flags:

CONSTANT

Example:

```
my int $v = $seq.getCurrent();
```

45.46.2.5 int Qore::Thread::Sequence::next ( )

Atomically increments the sequence value and returns the last value.

Returns

the last value of the sequence

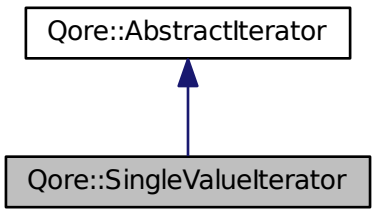
Example:

```
my int $next = $seq.next();
```

45.47 Qore::SingleValueIterator Class Reference

This class defines a simple iterator to be used to iterate single values (or complex objects where no iterator has been implemented yet)

Inheritance diagram for Qore::SingleValueIterator:



Public Member Functions

- constructor (any v)

  creates the single value iterator with the value passed as an argument
- copy ()

  Creates a copy of the SingleValueIterator object, iterating the same object as the original and in the same position.
- any getValue ()

  returns the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid
- bool next ()

  This method returns True and False alternately unless it has no value to iterate, in which case it returns only False.
- reset ()

  Reset the iterator instance to its initial state.
- bool valid ()

  returns True if the iterator is currently pointing at a valid element, False if not

45.47.1 Detailed Description

This class defines a simple iterator to be used to iterate single values (or complex objects where no iterator has been implemented yet)

Since

Qore 0.8.6

Example: SingleValueIterator basic usage

```
my any $val = 1;
my SingleValueIterator $it($val);
while ($it.next()) {
    printf("iter: %n\n", $it.getValue());
}

iter: 1
```

Remember that input value is taken as a single token so result of the code above for a list as an input argument will be like this:

```
my any $val = (1, 2, 3);
iter: list: (1, 2, 3)
```

45.47.2 Member Function Documentation

45.47.2.1 Qore::SingleValueIterator::constructor (any *v*)

creates the single value iterator with the value passed as an argument

Parameters

v	the value to iterate

Example:

```
my SingleValueIterator i($v);
```

45.47.2.2 Qore::SingleValueIterator::copy ( )

Creates a copy of the SingleValueIterator object, iterating the same object as the original and in the same position.

Example:

```
my SingleValueIterator $ni = $i.copy();
```

45.47.2.3 any Qore::SingleValueIterator::getValue ( ) [virtual]

returns the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Returns

the current value or throws an `INVALID-ITERATOR` exception if the iterator is invalid

Code Flags:

RET_VALUE_ONLY

Example:

```
while ($i.next()) {
    printf("+ %y\n", $i.getValue());
}
```

Exceptions

INVALID-ITERATOR	the iterator is not pointing at a valid element

Implements Qore::AbstractIterator.

45.47.2.4 bool Qore::SingleValueIterator::next ( ) [virtual]

This method returns True and False alternately unless it has no value to iterate, in which case it returns only False.

The iterator object should not be used after this method returns False

Returns

True and False alternately unless it has no value to iterate, in which case it returns only False

Example:

```
while ($i.next()) {
    printf("value: %y\n", $i.getValue());
}
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

Implements Qore::AbstractIterator.

45.47.2.5 Qore::SingleValueIterator::reset ( )

Reset the iterator instance to its initial state.

Reset the iterator instance to its initial state

Example

```
$i.reset();
```

Exceptions

ITERATOR-THREAD-ER↩ ROR	this exception is thrown if this method is called from any thread other than the thread that created the object

45.47.2.6 bool Qore::SingleValueIterator::valid ( ) [virtual]

returns True if the iterator is currently pointing at a valid element, False if not

Returns

True if the iterator is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

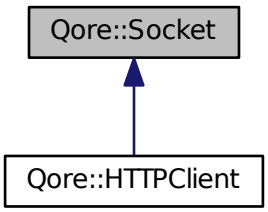
```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.48 Qore::Socket Class Reference

The Socket class allows Qore programs safe access to network sockets.

Inheritance diagram for Qore::Socket:



Public Member Functions

- Socket accept ()

  Accepts connections on a listening socket.
- __7_ Socket accept (timeout timeout_ms)

  Accepts connections on a listening socket (see Socket::listen()) accepting a timeout value with a millisecond resolution.
- Socket acceptSSL ()

  Accepts connections on a listening socket and attempts to negotiate a TLS/SSL connection.
- __7_ Socket acceptSSL (timeout timeout_ms)

  Accepts connections on a listening socket and attempts to negotiate a TLS/SSL connection accepting a timeout value with a millisecond resolution.
- int bind (string str, softbool reuseaddr=False)

  Opens and binds the socket to a port, interface and port (if the $bind_to string has a format "host:port"), or UNIX domain socket file.
- int bind (int port, softbool reuseaddr=False)

  Opens and binds the socket to an INET port on all interfaces.
- nothing bindINET (__7_ string iface, __7_ softstring service, softbool reuseaddr=False, softint family=AF_↩ UNSPEC, softint socktype=SOCK_STREAM, softint protocol=0)

  Opens and binds the socket to the given IPv4 or IPv6 interface (or if no interface is given, then to all interfaces on the local system) and port (the port number will be derived from the service name if a numeric port number is not given)
- nothing bindUNIX (string path, softint socktype=SOCK_STREAM, softint protocol=0)

  Opens and binds the socket to the given UNIX domain socket file as given by the filename argument. If any errors occur, an exception is thrown.
- clearStats ()

  Clears performance statistics.
- nothing clearWarningQueue ()

  Removes any warning Queue object from the Socket.
- int close ()

  Closes an open socket.
- nothing connect (string target, timeout timeout_ms=-1)

  Connects to a remote port (if the string has a format "host:port") or UNIX domain socket file with an optional timeout value with a millisecond resolution.
- nothing connectINET (string host, softstring service, timeout timeout_ms=-1, softint family=AF_UNSPEC, softint socktype=SOCK_STREAM, softint protocol=0)

  Connects to the given host and port with an optional timeout value with a millisecond resolution.

- nothing connectINETSSL (string host, softstring service, timeout timeout_ms=-1, softint family=AF_UNSPEC, softint socktype=SOCK_STREAM, softint protocol=0)

  Connects to the given host and port and attempts to establish a TLS/SSL connection; accepts an optional timeout value with a millisecond resolution.
- nothing connectSSL (string target, timeout timeout_ms=-1)

  Connects to a remote socket and attempts to establish a TLS/SSL connection; accepts an optional timeout value with a millisecond resolution.
- nothing connectUNIX (string path, softint socktype=SOCK_STREAM, softint protocol=0)

  Connects to a UNIX domain socket file.
- nothing connectUNIXSSL (string path, softint socktype=SOCK_STREAM, softint protocol=0)

  Connects to the given UNIX domain socket file and attempts to establish a TLS/SSL connection.
- constructor ()

  Creates the socket object.
- copy ()

  Creates a new Socket object, not based on the source being copied.
- string getCharset ()

  Returns the character encoding for the socket.
- string getEncoding ()

  Returns the character encoding for the socket.
- bool getNoDelay ()

  Returns the `TCP_NODELAY` setting for the socket.
- hash getPeerInfo (bool host_lookup=True)

  Returns a hash of information about the remote end for connected sockets.
- int getPort ()

  Returns the port number of the socket for INET sockets.
- int getRecvTimeout ()

  Returns the receive timeout socket option value as an integer in milliseconds.
- __7_ string getSSLCipherName ()

  Returns the name of the cipher for an encrypted connection or NOTHING if a secure connection has not been established.
- __7_ string getSSLCipherVersion ()

  Returns the version string of the cipher for an encrypted connection or NOTHING if a secure connection has not been established.
- int getSendTimeout ()

  Returns the send timeout socket option value as an integer in milliseconds.
- int getSocket ()

  Returns the socket file descriptor number.
- hash getSocketInfo (bool host_lookup=True)

  Returns information about the local socket as a hash.
- hash getUsageInfo ()

  Returns performance statistics for the socket.
- bool isDataAvailable (timeout timeout_ms=0)

  Returns True or False depending on whether there is data to be read on the socket.
- bool isOpen ()

  Returns True if the socket is open.
- bool isSecure ()

  Returns True if the connection is a secure TLS/SSL connection.
- bool isWriteFinished (timeout timeout_ms=0)

  Returns True or False depending on whether all the data has been written to the socket.
- int listen (int backlog=20)

  Listens for connections on a bound socket; sets the socket in a listening state.
- bool pendingHttpChunkedBody ()

returns True if the socket is still connected, and a HTTP header was read indicating chunked transfer encoding, but no chunked body has been read yet

- hash readHTTPChunkedBody (timeout timeout_ms=-1)

  Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the "`body`" key of a hash (including footers received)
- hash readHTTPChunkedBodyBinary (timeout timeout_ms=-1)

  Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a binary object in the "`body`" key of a hash (including footers received)
- readHTTPChunkedBodyBinaryWithCallback (code rcb, timeout timeout_ms=-1)

  Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the "`body`" key of a hash (including footers received)
- readHTTPChunkedBodyWithCallback (code rcb, timeout timeout_ms=-1)

  Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the "`body`" key of a hash (including footers received)
- hash readHTTPHeader (timeout timeout_ms=-1, __7_ reference info)

  Retuns a hash representing the data in the HTTP header read, or, if the data cannot be parsed as an HTTP header, then an exception is thrown, and the data read is returned as a string in the `arg` key of the exception hash.
- string readHTTPHeaderString (timeout timeout_ms=-1)

  Retuns a string representing the data in the HTTP header read (reads until "`\r\n\r\n`")
- string recv (softint size=0, timeout timeout_ms=-1)

  Receives data from the socket and returns a string tagged with the Socket's character encoding.
- binary recvBinary (softint size=0, timeout timeout_ms=-1)

  Receives data from the socket and returns a binary object.
- int recvi1 (timeout timeout_ms=-1)

  Receives a 1-byte signed integer from the socket.
- int recvi2 (timeout timeout_ms=-1)

  Receives a 2-byte (16-bit) signed integer in big-endian format (network byte order) from the socket.
- int recvi2LSB (timeout timeout_ms=-1)

  Receives a 2-byte (16-bit) signed integer in little-endian format from the socket.
- int recvi4 (timeout timeout_ms=-1)

  Receives a 4-byte (32-bit) signed integer in big-endian format (network byte order) from the socket.
- int recvi4LSB (timeout timeout_ms=-1)

  Receives a 4-byte (32-bit) signed integer in little-endian format from the socket.
- int recvi8 (timeout timeout_ms=-1)

  Receives an 8-byte (64-bit) signed integer in big-endian format (network byte order) from the socket.
- int recvi8LSB (timeout timeout_ms=-1)

  Receives an 8-byte (64-bit) signed integer in little-endian format from the socket.
- int recvu1 (timeout timeout_ms=-1)

  Receives a 1-byte unsigned integer from the socket.
- int recvu2 (timeout timeout_ms=-1)

  Receives a 2-byte (16-bit) unsigned integer in big-endian format (network byte order) from the socket.
- int recvu2LSB (timeout timeout_ms=-1)

  Receives a 2-byte (16-bit) unsigned integer in little-endian format from the socket.
- int recvu4 (timeout timeout_ms=-1)

  Receives a 4-byte (32-bit) unsigned integer in big-endian format (network byte order) from the socket.
- int recvu4LSB (timeout timeout_ms=-1)

  Receives a 4-byte (32-bit) unsigned integer in little-endian format from the socket.
- int send (binary bin, int timeout_ms=-1)

  Sends binary data over the socket; if any errors occur, an exception is thrown.
- int send (string str, timeout timeout_ms=-1)

  Sends string data over the socket; string data is converted to the socket's encoding if necessary; if any errors occur, an exception is thrown.

- nothing send2 (binary bin, timeout timeout_ms=-1)

  Sends binary data over the socket; if any errors occur, an exception is thrown.
- nothing send2 (string str, timeout timeout_ms=-1)

  Sends string data over the socket; string data is converted to the socket's encoding if necessary; if any errors occur, an exception is thrown.
- int sendBinary (string str, timeout timeout_ms=-1)

  Sends string data over the socket without converting the string to the socket's encoding, but instead is sent exactly as-is; if any errors occur, an exception is thrown.
- int sendBinary (binary bin, timeout timeout_ms=-1)

  Sends binary data over the socket; if any errors occur, an exception is thrown.
- nothing sendBinary2 (string str, timeout timeout_ms=-1)

  Sends string data over the socket without converting the string to the socket's encoding, but instead is sent exactly as-is; if any errors occur, an exception is thrown.
- nothing sendBinary2 (binary bin, timeout timeout_ms=-1)

  Sends binary data over the socket; if any errors occur, an exception is thrown.
- nothing sendHTTPMessage (string method, string path, string http_version, hash headers, __7_ string body, __7_ reference info, timeout timeout_ms=-1)

  Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.
- nothing sendHTTPMessage (string method, string path, string http_version, hash headers, binary body, __↩ 7_ reference info, timeout timeout_ms=-1)

  Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.
- nothing sendHTTPMessageWithCallback (code scb, string method, string path, string http_version, hash headers, __7_ reference info, timeout timeout_ms=-1)

  Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.
- nothing sendHTTPResponse (softint status_code, string status_desc, string http_version, hash headers, _↩ _7_ string body, timeout timeout_ms=-1)

  Sends an HTTP response with user-defined headers given as a hash and an optional message body.
- nothing sendHTTPResponse (softint status_code, string status_desc, string http_version, hash headers, binary body, timeout timeout_ms=-1)

  Sends an HTTP response with user-defined headers given as a hash and a message body as literal binary data.
- nothing sendHTTPResponseWithCallback (code scb, softint status_code, string status_desc, string http_↩ version, hash headers, timeout timeout_ms=-1)

  Sends an HTTP response with user-defined headers given as a hash and a message body as literal binary data.
- int sendi1 (softint i=0, timeout timeout_ms=-1)

  Sends a 1-byte integer over the socket.
- int sendi2 (softint i=0, timeout timeout_ms=-1)

  Sends a 2-byte (16-bit) integer in big-endian format (network byte order) over the socket.
- int sendi2LSB (softint i=0, timeout timeout_ms=-1)

  Sends a 2-byte (16-bit) integer in little-endian format over the socket.
- int sendi4 (softint i=0, timeout timeout_ms=-1)

  Sends a 4-byte (32-bit) integer in big-endian format (network byte order) over the socket.
- int sendi4LSB (softint i=0, timeout timeout_ms=-1)

  Sends a 4-byte (32-bit) integer in little-endian format over the socket.
- int sendi8 (softint i=0, timeout timeout_ms=-1)

  Sends an 8-byte (64-bit) integer in big-endian format (network byte order) over the socket.
- int sendi8LSB (softint i=0, timeout timeout_ms=-1)

  Sends an 8-byte (64-bit) integer in little-endian format over the socket.
- nothing setCertificate (SSLCertificate cert)

  Sets the X.509 certificate to use for negotiating encrypted connections.
- nothing setCertificate (string cert_pem)

  Sets the X.509 certificate to use for negotiating encrypted connections from the PEM-encoded string representing the X.509 certificate.

- nothing setCertificate (binary cert_der)

  Sets the X.509 certificate to use for negotiating encrypted connections from the DER-encoded binary object representing the X.509 certificate.
- nothing setCharset (string encoding)

  Sets the character encoding for the socket.
- nothing setEncoding (string encoding)

  Sets the character encoding for the socket.
- nothing setEventQueue ()

  Removes any Queue object from the Socket object so that socket events are no longer added to the Queue.
- nothing setEventQueue (Queue queue)

  Sets a Queue object to receive socket events.
- int setNoDelay (bool nd=True)

  Sets the boolean `TCP_NODELAY` setting for the socket.
- nothing setPrivateKey (SSLPrivateKey key)

  Sets the private key to use for negotiating encrypted connections along with the X.509 certificate.
- nothing setPrivateKey (string key_pem, __7_ string pass)

  Sets the private key to use for negotiating encrypted connections along with the X.509 certificate from a PEM-encoded string representing the private key and an optional password.
- nothing setPrivateKey (binary key_der)

  Sets the private key to use for negotiating encrypted connections along with the X.509 certificate from a DER-encoded binary object representing the private key.
- int setRecvTimeout (timeout timeout_ms)

  sets the receive timeout as a socket option
- int setSendTimeout (timeout timeout_ms)

  sets the send timeout as a socket option
- nothing setWarningQueue (int warning_ms, int warning_bs, Queue queue, any arg, timeout min_ms=1s)

  Sets a Queue object to receive socket warnings.
- int shutdown ()

  Ensures that a socket will be closed even if the file descriptor is shared with other processes (for example, after a call to fork())
- nothing shutdownSSL ()

  Shuts down the SSL connection on a secure connection.
- nothing upgradeClientToSSL ()

  Upgrades a client socket connection to a TLS/SSL connection.
- nothing upgradeServerToSSL ()

  Upgrades a server socket connection to a TLS/SSL connection.
- __7_ string verifyPeerCertificate ()

  Returns a string code giving the result of verifying the remote certificate or NOTHING if an encrypted connection is not currently established.

45.48.1 Detailed Description

The Socket class allows Qore programs safe access to network sockets.

Restrictions:

Qore::PO_NO_NETWORK

Note

This class is not available with the PO_NO_NETWORK parse option.

Non-blocking socket I/O can be performed by appending a timeout value in milliseconds all Socket methods performing I/O operations, or by using the Socket::isDataAvailable() method with a timeout value in milliseconds (1000 ms = 1 second). Note that as with all Qore functions and methods accepting a timeout value, relative date/time values can be given instead of integers to make the source more readable, for example:

```
my bool $rc = $socket.isDataAvailable(1250ms); # times out in 1.25 seconds
```

Socket objects can automatically convert character encodings if desired when sending string data with Socket::send(). Use the Socket::setEncoding() method to set the character encoding for the socket. If a character encoding is set, and string data is read with the Socket::recv() method, then it will be tagged with the encoding of the socket as well.

Client applications should call Socket::connect() to connect to a remote port or a UNIX domain socket (socket file on the local server). However, if the remote end is expecting a TLS/SSL connection, use Socket::connectSSL() instead.

Server applications should call Socket::bind(), Socket::listen(), and Socket::accept() in this order to accept incoming connections. Normally a new thread should be started after the Socket::accept() call to handle the new connection in a separate thread (Socket::accept() returns a new Socket object for the accepted connection).

To support TLS/SSL server connections, first set the certificate and private key with the Socket::setCertificate() and Socket::setPrivateKey() methods (see the SSLCertificate Class and the SSLPrivateKey Class for more information on the parameters required for these methods). Then Socket::acceptSSL() should be called after the socket is in a listening state to accept client connections and negotiate a TLS/SSL connection.

This class supports posting events to a Queue. See I/O Event Handling for more information.

The events raised by this object are listed in the following table:

Socket Events

Name	Description
EVENT_PACKET_READ	Raised when a network packet is received.
EVENT_PACKET_SENT	Raised when a network packet is sent.
EVENT_CHANNEL_CLOSED	Raised when a socket is closed.
EVENT_DELETED	Raised when the object being monitored is deleted.
EVENT_HOSTNAME_LOOKUP	Raised when a hostname lookup is attempted.
EVENT_HOSTNAME_RESOLVED	Raised when a hostname lookup is resolved.
EVENT_HTTP_SEND_MESSAGE	Raised when an HTTP message is sent.
EVENT_HTTP_MESSAGE_RECEIVED	Raised when an HTTP message is received.
EVENT_CONNECTING	Raised right before a socket connection attempt is made.
EVENT_CONNECTED	Raised when the socket connection has been established.
EVENT_START_SSL	Raised when socket SSL negotiation starts.
EVENT_SSL_ESTABLISHED	Raised when SSL communication has been negotiated and established.

Socket Information Hash

Key	Description
`hostname`	The interface name if available (ex: `"localhost"`; note that this key is not present when retrieving information about UNIX sockets)

hostname_desc	A descriptive string giving the hostname and the address family if the hostname is available (ex: "ipv6[localhost]"; note that this key is not present when retrieving information about UNIX sockets)
address	A string giving the address (ex: "::ffff:0.0.0.0")
address_desc	A descriptive string giving the address and the address family (ex: "ipv6[::ffff:0.0.0.0]")
port	An integer port number if available (note that this key is not present when retrieving information about UNIX sockets)
family	The network address family (see Network Address Family Constants)
familystr	A string describing the network address family (ex: "ipv4")

45.48.2 Member Function Documentation

45.48.2.1 Socket Qore::Socket::accept ( )

Accepts connections on a listening socket.

Accepts connections on a listening socket; if any errors occur, an exception is thrown.

The new Socket object returned will have the same character encoding as the current object. Once a new connection has been accepted, call Socket::getPeerInfo() to get information about the remote socket.

Example:

```
my Socket $new_socket = $sock.accept();
```

Returns

a new Socket object is returned for the new connection

Exceptions

SOCKET-NOT-OPEN	The socket is not bound
SOCKET-ACCEPT-ERROR	Error in accepting connection

See also

Socket::acceptSSL(), Socket::listen(), Socket::getPeerInfo()

45.48.2.2 __7_ Socket Qore::Socket::accept (timeout *timeout_ms*)

Accepts connections on a listening socket (see Socket::listen()) accepting a timeout value with a millisecond resolution.

If any errors occur, an exception is thrown.

The new Socket object returned will have the same character encoding as the current object. Once a new connection has been accepted, call Socket::getPeerInfo() to get information about the remote socket.

Example:

```
my *Socket $new_socket = $sock.accept(30s);
```

Parameters

timeout_ms	If a timeout value is passed and the connection takes longer to establish than the timeout, then NOTHING is returned.Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

If no connection is accepted within the timeout period, then NOTHING is returned, otherwise a Socket object for the new connection is returned.

Exceptions

SOCKET-NOT-OPEN	The socket is not bound
SOCKET-ACCEPT-ERROR	Error in accepting connection

See also

Socket::acceptSSL(), Socket::listen(), Socket::getPeerInfo()

45.48.2.3 Socket Qore::Socket::acceptSSL ( )

Accepts connections on a listening socket and attempts to negotiate a TLS/SSL connection.

Accepts connections on a listening socket and attempts to negotiate a TLS/SSL connection; if any errors occur, an exception is thrown.

The new Socket object returned will have the same character encoding as the current object. Once a new connection has been accepted, call Socket::getPeerInfo() to get information about the remote socket.

Example:

```
my Socket $new_socket = $sock.acceptSSL();
```

Returns

When a new connection is accepted and a TLS/SSL session has been successfully negotiated, a new Socket object is returned for the new connectio

Exceptions

SOCKET-NOT-OPEN	The socket is not bound
SOCKET-ACCEPT-ERROR	Error in accepting connection

See also

Socket::accept(), Socket::listen(), Socket::getPeerInfo()

45.48.2.4 __7_ Socket Qore::Socket::acceptSSL (timeout *timeout_ms*)

Accepts connections on a listening socket and attempts to negotiate a TLS/SSL connection accepting a timeout value with a millisecond resolution.

If any errors occur, an exception is thrown.

The new Socket object returned will have the same character encoding as the current object. Once a new connection has been accepted, call Socket::getPeerInfo() to get information about the remote socket.

Example:

```
my *Socket $new_socket = $sock.acceptSSL(30s);
```

Parameters

timeout_ms	If a timeout value is passed and the connection takes longer to establish than the timeout, then NOTHING is returned. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

If no connection is accepted within the timeout period, then NOTHING is returned, otherwise a Socket object for the new connection is returned.

Exceptions

SOCKET-NOT-OPEN	The socket is not bound
SOCKET-ACCEPT-ERROR	Error in accepting connection

See also

Socket::acceptSSL(), Socket::listen(), Socket::getPeerInfo()

45.48.2.5 int Qore::Socket::bind (string *str*, softbool *reuseaddr* = False)

Opens and binds the socket to a port, interface and port (if the $bind_to string has a format "host:port"), or UNIX domain socket file.

If the second parameter is True, then the socket will set the `SO_REUSEADDR` option, which will allow the socket to be bound to a port that is not yet closed (for example, in a `TIME_WAIT` state).

If any errors occur a non-zero error code is returned.

This method tries to automatically pick the appropriate address family from the arguments; note that a hostname or address in square brackets (ex: `"[localhost]"`) will be looked up and bound as an IPv6 address; additionally, the method recognizes ipv6 addresses by embedded colons (:) in the address string and binds them as such.

Internally, the getaddrinfo() function is used to look up bind addresses; internal `bind()` operations are tried in sequence for each address returned; as soon as a bind operation is successful, the method returns. If none of the addresses can be bound, then an error code is returned.

Example:

```
 # bind to a UNIX socket on the local system and reuse the address, check return code
if ($sock.bind("/tmp/my-socket", True))
    throw "BIND-ERROR", strerror();
```

Parameters

str	If a colon appears in the string, the string will be assumed to be a `"bind_address:port"` specification, and the port on the named IP address will be bound, otherwise, if the string contains no colon, the socket will be bound to a UNIX domain socket file on the local filesystem with the given name. Note that a hostname or address in square brackets (ex: `"[localhost]"`) will be looked up and bound as an IPv6 address; additionally, the method recognizes ipv6 addresses by embedded colons (:) in the address string (if surrounded by square brackets) and binds them as such.

reuseaddr	If this optional argument evaluates to True, the SO_REUSEADDR option will be set on the socket, which will allow the socket to be bound to a port that is not yet closed (for example, in a TIME_WAIT state); note that this only applies to IPv4 (AF_INET) and IPv6 (AF_INET6) sockets; this option is ignored for UNIX (AF_UNIX) sockets.

See also

Socket::bindINET() and Socket::bindUNIX()

Note

UNIX domain sockets are not available on native Windows ports

45.48.2.6 int Qore::Socket::bind (int *port*, softbool *reuseaddr* = False)

Opens and binds the socket to an INET port on all interfaces.

If the second parameter is True, then the socket will set the SO_REUSEADDR option, which will allow the socket to be bound to a port that is not yet closed (for example, in a TIME_WAIT state).

If any errors occur a non-zero error code is returned.

Example:

```
 # bind to port 80 on all interfaces on the local system and reuse the address, check return code
if ($sock.bind(80, True))
    throw "BIND-ERROR", strerror();
```

Parameters

port	A port number to bind to on all interfaces
reuseaddr	If this optional argument evaluates to True, the SO_REUSEADDR option will be set on the socket, which will allow the socket to be bound to a port that is not yet closed (for example, in a TIME_WAIT state)

See also

Socket::bindINET() and Socket::bindUNIX()

45.48.2.7 nothing Qore::Socket::bindINET (__7_ string *iface*, __7_ softstring *service*, softbool *reuseaddr* = False, softint *family* = AF_UNSPEC, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Opens and binds the socket to the given IPv4 or IPv6 interface (or if no interface is given, then to all interfaces on the local system) and port (the port number will be derived from the service name if a numeric port number is not given)

Opens and binds the socket to a port, interface and port (if the interface string has a format "host:port"), or UNIX domain socket file (if no port or service name appears in the bind string). If the second parameter is True, then the socket will set the SO_REUSEADDR option, which will allow the socket to be bound to a port that is not yet closed (for example, in a TIME_WAIT state).

Internally, the getaddrinfo() function is used to look up bind addresses; internal bind() operations are tried in sequence for each address returned; as soon as a bind operation is successful, the method returns. If none of the addresses can be bound, then an error code is returned.

If any errors occur, an exception is thrown.

Examples:

```
 # bind to port 80 on all interfaces on the local system and reuse the address
$sock.bindINET(NOTHING, 80, True);
```

```
 # bind to interface 192.168.2.23 port 8080 and do not reuse the address
$sock.bindINET("192.168.2.23", 8080);

 # bind to localhost port 8080 with ipv6 and do not reuse the address
$sock.bindINET("localhost", 8080, False, AF_INET6);

 # bind to ipv6 host address fe80::21c:42ff:fe00:8, port 1001, reuse the address
$sock.bindINET("fe80::21c:42ff:fe00:8", 1001, True);
```

Parameters

iface	the host name or IP address to bind to
service	the service name (ex: "http" or port number (given as or converted to a string) to bind to
reuseaddr	if this optional argument evaluates to True, the SO_REUSEADDR option will be set on the socket, which will allow the socket to be bound to a port that is not yet closed (for example, in a TIME_WAIT state)
family	the address family to use to bind; see Network Address Family Constants; AF_UNSPEC means to try all possible addres families
socktype	the type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	the protocol number for the socket; use 0 for the default protocol

Exceptions

SOCKET-BIND-ERROR	Both hostname and service name are empty or not set; error opening socket for bind; error binding on socket.
QOREADDRINFO-GETINFO-ERROR	error looking up either nodename or servicename (or not known)

See also

Socket::bind() and Socket::bindUNIX()

45.48.2.8 nothing Qore::Socket::bindUNIX (string *path*, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Opens and binds the socket to the given UNIX domain socket file as given by the filename argument. If any errors occur, an exception is thrown.

Opens and binds the socket to the given UNIX domain socket file as given by the filename argument. Note that the socket file is automatically deleted in the destructor when a UNIX socket is closed. If any errors occur, an exception is thrown.

Example:

```
 # bind to UNIX domain socket file "/tmp/socket"
$sock.bindUNIX("/tmp/socket");
```

Parameters

path	The path of the UNIX domain socket to create and bind to
socktype	the type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	the protocol number for the socket; use 0 for the default protocol

Exceptions

SOCKET-BIND-ERROR	Error opening socket for bind; error binding on socket

See also

Socket::bind() and Socket::bindINET()

Note

UNIX domain sockets are not available on native Windows ports

45.48.2.9 Qore::Socket::clearStats ( )

Clears performance statistics.

Example:

```
$sock.clearStats();
```

Since

Qore 0.8.9

See also

Socket::getUsageInfo()

45.48.2.10 nothing Qore::Socket::clearWarningQueue ( )

Removes any warning Queue object from the Socket.

Example:

```
$sock.clearWarningQueue();
```

See also

Socket::setWarningQueue()

Since

Qore 0.8.9

45.48.2.11 int Qore::Socket::close ( )

Closes an open socket.

Also deletes the UNIX domain socket file if it was created by a call to Socket::bind(). Returns 0 for success, -1 for error; in this case check errno() for the error number

Example:

```
if ($sock.close())
    stderr.printf("Error closing socket: %s\n", strerror(errno()));
```

Events:

EVENT_CHANNEL_CLOSED

Returns

0 for success, -1 for error; in this case check errno() for the error number

45.48.2.12 nothing Qore::Socket::connect (string *target*, timeout *timeout_ms* = -1)

Connects to a remote port (if the string has a format "host:port") or UNIX domain socket file with an optional timeout value with a millisecond resolution.

Connects the socket to a remote (or local) port or UNIX domain socket file, for network (ipv4 and ipv6) connections, accepts an optional timeout value in milliseconds (relative date/time values can be given instead of an integer in milliseconds to make the source more readable; ex: `20s`). If any errors occur, an exception is thrown.

Examples:

```
 # connect to ipv4 adress 192.168.1.45 port 8080 with a 30 second timeout
$sock.connect("192.168.1.45:8080", 30s);

 # connect to ipv6 adress 2001:0db8:85a3:0000:0000:8a2e:0370:7334 port 80 with a 10 second timeout
$sock.connect("[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:80", 10s);

 # connect to localhost using ipv6 (::1) port 80 with a 10 second timeout
$sock.connect("[localhost]:80", 15s);

 # connect to UNIX domain socket file "/tmp/socket"
$sock.connect("/tmp/socket");
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED

Parameters

target	The target address interpreted with the following rules: • If a colon appears in the string, the string will be assumed to be a "`hostname:port`" specification to connect to; where "`port`" can also be a service name (ex: "`example.com:https`") • If the string contains no colon, the socket will try to connect to a UNIX domain socket file on the local filesystem with the given name. • Enclose ipv6 addresses in square brackets (ex: "`[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:80`", will connect to port 80 on the given ipv6 address); also if a hostname is enclosed in square brackets, it will be resolved as an ipv6 address (ex: "`[localhost]:80`" will connect to port `80` on ::1, assuming "`localhost`" can be resolved to the ipv6 loopback address on the system).

timeout_ms	If a timeout value is passed and the connection takes longer to establish than the timeout, a "SOCKET-CONNECT-ERROR" exception is thrown. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Exceptions

SOCKET-CONNECT-ERR↩ OR	An error occured connecting to the remote socket (cannot resolve hostname, no listener, timeout exceeded, etc).

See also

Socket::connectUNIX(), Socket::connectUNIX(), Socket::connectSSL(), Socket::connectINETSSL(), and Socket::connectUNIXSSL()

45.48.2.13 nothing Qore::Socket::connectINET (string *host*, softstring *service*, timeout *timeout_ms* = -1, softint *family* = AF_UNSPEC, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Connects to the given host and port with an optional timeout value with a millisecond resolution.

Connects the socket to a remote (or local) port; accepts an optional timeout value in milliseconds (relative date/time values can be given instead of an integer in milliseconds to make the source more readable; ex: 20s). If any errors occur, an exception is thrown.

Do not use square brackets to designate ipv6 addresses with this method; just give the address in its normal form (ex: "2001:0db8:85a3:0000:0000:8a2e:0370:7334").

Examples:

```
 # connect to ipv4 address 192.168.1.45 port 8080 with a 30 second timeout
$sock.connectINET("192.168.1.45", 8080, 30s);

 # connect to ipv6 address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 port 80 with a 20 second timeout
$sock.connectINET("2001:0db8:85a3:0000:0000:8a2e:0370:7334", 80, 20s);
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_R↩ ESOLVED

Parameters

host	The host name or IP address to connect to
service	The service name (ex: "http" or port number (given as or converted to a string) to connect to
timeout_ms	If a timeout value is passed and the connection takes longer to establish than the timeout, a "SOCKET-CONNECT-ERROR" exception is thrown. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)
family	The address family to use to connect to the remote socket; see Network Address Family Constants
socktype	The type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	The protocol number for the socket (if not sure leave this 0)

Exceptions

SOCKET-CONNECT-ERR↩ *OR*	An error occured connecting to the remote socket (cannot resolve hostname, no listener, timeout exceeded, etc)

See also

Socket::connect(), Socket::connectUNIX(), Socket::connectSSL(), Socket::connectINETSSL(), and Socket↩ ::connectUNIXSSL()

45.48.2.14 nothing Qore::Socket::connectINETSSL (string *host*, softstring *service*, timeout *timeout_ms* = -1, softint *family* = AF_UNSPEC, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Connects to the given host and port and attempts to establish a TLS/SSL connection; accepts an optional timeout value with a millisecond resolution.

Connects the socket to a remote (or local) port and attempts to establish a TLS/SSL connection; accepts an optional timeout value in milliseconds (relative date/time values can be given instead of an integer in milliseconds to make the source more readable; ex: `20s`). If any errors occur, an exception is thrown.

Do not use square brackets to designate ipv6 addresses with this method; just give the address in its normal form (ex: `"2001:0db8:85a3:0000:0000:8a2e:0370:7334"`).

Examples:

```
 # connect to 192.168.1.45 port 8080 with a 30 second timeout
$sock.connectINETSSL("192.168.1.45", 8080, 30s);

 # connect to ipv6 address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 port 80 with a 20 second timeout
$sock.connectINETSSL("2001:0db8:85a3:0000:0000:8a2e:0370:7334", 80, 20s);
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_R↩ ESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED

Parameters

host	The host name or IP address to connect to
service	The service name (ex: `"http"` or port number (given as or converted to a string) to connect to
timeout_ms	If a timeout value is passed and the connection takes longer to establish than the timeout, a `"SOCKET-CONNECT-ERROR"` exception is thrown. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)
family	The address family to use to connect to the remote socket; see Network Address Family Constants
socktype	The type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	The protocol number for the socket (if not sure leave this 0)

Exceptions

SOCKET-CONNECT-ERR↩ *OR*	An error occured connecting to the remote socket (cannot resolve hostname, no listener, timeout exceeded, etc)
SOCKET-SSL-ERROR	An error occurred establishing the TLS/SSL connection

See also

Socket::connect(), Socket::connectUNIX(), Socket::connectSSL(), Socket::connectINET(), and Socket↩ ::connectUNIXSSL()

45.48.2.15 nothing Qore::Socket::connectSSL (string *target*, timeout *timeout_ms* = -1)

Connects to a remote socket and attempts to establish a TLS/SSL connection; accepts an optional timeout value with a millisecond resolution.

Connects to a remote socket and attempts to establish a TLS/SSL connection, for network (INET) connections, accepts an optional timeout value in milliseconds (relative date/time values can be given instead of an integer in milliseconds to make the source more readable; ex: `20s`). If any errors occur, an exception is thrown.

Enclose ipv6 addresses in square brackets (ex: `"[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:80"`, will connect to port 80 on the given ipv6 address); also if a hostname is enclosed in square brackets, it will be resolved as an ipv6 adress (ex: `"[localhost]:80"` will connect to port `80` on ::1, assuming `"localhost"` can be resolved to the ipv6 loopback address on the system).

Examples:

```
 # connect to ipv4 adress 192.168.1.45 port 8080 with a 30 second timeout
$sock.connectSSL("192.168.1.45:8080", 30s);

 # connect to ipv6 adress 2001:0db8:85a3:0000:0000:8a2e:0370:7334 port 80 with a 10 second timeout
$sock.connectSSL("[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:80", 10s);

 # connect to localhost using ipv6 (::1) port 80 with a 10 second timeout
$sock.connectSSL("[localhost]:80", 15s);

 # connect to UNIX domain socket file "/tmp/socket"
$sock.connectSSL("/tmp/socket");
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_HOSTNAME_LOOKUP, EVENT_HOSTNAME_RESOLVED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED

Parameters

target	The target address interpreted with the following rules: • If a colon appears in the string, the string will be assumed to be a `"hostname:port"` specification to connect to; where `"port"` can also be a service name (ex: `"example.com:https"`) • If the string contains no colon, the socket will try to connect to a UNIX domain socket file on the local filesystem with the given name. • Enclose ipv6 addresses in square brackets (ex: `"[2001:0db8:85a3:0000:0000:8a2e:0370:7334]:80"`, will connect to port 80 on the given ipv6 address); also if a hostname is enclosed in square brackets, it will be resolved as an ipv6 address (ex: `"[localhost]:80"` will connect to port `80` on ::1, assuming `"localhost"` can be resolved to the ipv6 loopback address on the system).

timeout_ms	if a timeout value is passed and the connection takes longer to establish than the timeout, an exception is thrown. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-CONNECT-ERR↩ *OR*	An error occured connecting to the remote socket (cannot resolve hostname, no listener, timeout exceeded, etc)
SOCKET-SSL-ERROR	An error occurred establishing the TLS/SSL connection

See also

Socket::connect(), Socket::connectUNIX(), Socket::connectUNIXSSL(), Socket::connectINETSSL(), and Socket::connectINET()

45.48.2.16 nothing Qore::Socket::connectUNIX (string *path*, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Connects to a UNIX domain socket file.

Connects the socket to the given UNIX domain socket file; if any errors occur, an exception is thrown

Example:

```
 # connect to UNIX domain socket file "/tmp/socket"
$sock.connectUNIX("/tmp/socket");
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED

Parameters

path	The socket will try to connect to a UNIX domain socket file on the local filesystem with the given name
socktype	The type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	The protocol number for the socket (if not sure leave this 0)

Exceptions

SOCKET-CONNECT-ERR↩ *OR*	An error occured connecting to the socket

See also

Socket::connect(), Socket::connectINET(), Socket::connectSSL(), Socket::connectINETSSL(), and Socket↩ ::connectUNIXSSL()

Note

UNIX domain sockets are not available on native Windows ports

45.48.2.17 nothing Qore::Socket::connectUNIXSSL (string *path*, softint *socktype* = SOCK_STREAM, softint *protocol* = 0)

Connects to the given UNIX domain socket file and attempts to establish a TLS/SSL connection.

Connects the socket to a UNIX domain socket file and attempts to establish a TLS/SSL connection. If any errors occur, an exception is thrown.

Example:

```
 # connect to UNIX domain socket file "/tmp/socket"
$sock.connectUNIXSSL("/tmp/socket");
```

Events:

EVENT_CONNECTING, EVENT_CONNECTED, EVENT_START_SSL, EVENT_SSL_ESTABLISHED

Parameters

path	The socket will try to connect to a UNIX domain socket file on the local filesystem with the given name
socktype	The type of socket; see Socket Type Constants; typically SOCK_STREAM for TCP sockets
protocol	The protocol number for the socket (if not sure leave this 0)

Exceptions

SOCKET-CONNECT-ERROR	An error occured connecting to the socket
SOCKET-SSL-ERROR	An error occurred establishing the TLS/SSL connection

See also

Socket::connect(), Socket::connectUNIX(), Socket::connectSSL(), Socket::connectINETSSL(), and Socket::connectINET()

Note

UNIX domain sockets are not available on native Windows ports

45.48.2.18 Qore::Socket::constructor ( )

Creates the socket object.

Example:

```
my Socket $sock();
```

45.48.2.19 Qore::Socket::copy ( )

Creates a new Socket object, not based on the source being copied.

Example:

```
my Socket $newsock = $sock.copy();
```

45.48.2.20 string Qore::Socket::getCharset ( )

Returns the character encoding for the socket.

Code Flags:

CONSTANT

A method synonym for Socket::getEncoding(), kept for backwards-compatibility

Returns

the character encoding for the socket

45.48.2.21 string Qore::Socket::getEncoding ( )

Returns the character encoding for the socket.

Code Flags:

CONSTANT

Returns

the character encoding for the socket

45.48.2.22 bool Qore::Socket::getNoDelay ( )

Returns the `TCP_NODELAY` setting for the socket.

Code Flags:

CONSTANT

Example:

```
my bool $b = $sock.getNoDelay();
```

Returns

the `TCP_NODELAY` setting for the socket

See also

Socket::setNoDelay() for a description of the `TCP_NODELAY` setting

45.48.2.23 hash Qore::Socket::getPeerInfo (bool *host_lookup* = True)

Returns a hash of information about the remote end for connected sockets.

If the socket is not connected, an exception is thrown

Example:

```
my hash $h = $sock.getPeerInfo();
```

Parameters

host_lookup	do a lookup of the hostname; if this is False then `"hostname"` and `"hostname_desc"` are not present in the response hash

Returns

a hash of information about the remote end for connected sockets

Exceptions

SOCKET-GETPEERINFO-ERROR	Socket is not open or error in getpeername()

Since

Qore 0.8.8 added the *host_lookup* parameter

45.48.2.24 int Qore::Socket::getPort ( )

Returns the port number of the socket for INET sockets.

Code Flags:

CONSTANT

Example:

```
my int $port = $sock.getPort();
```

Returns

the port number for an INET connection, -1 if no INET connection has been established

45.48.2.25 int Qore::Socket::getRecvTimeout ( )

Returns the receive timeout socket option value as an integer in milliseconds.

Code Flags:

CONSTANT

Example:

```
my int $to = $sock.getRecvTimeout();
```

Returns

the receive timeout socket option value as an integer in milliseconds

45.48.2.26 int Qore::Socket::getSendTimeout ( )

Returns the send timeout socket option value as an integer in milliseconds.

Code Flags:

CONSTANT

Example:

```
my int $to = $sock.getSendTimeout();
```

Returns

the send timeout socket option value as an integer in milliseconds

45.48.2.27 int Qore::Socket::getSocket ( )

Returns the socket file descriptor number.

Code Flags:

CONSTANT

Example:

```
my int $sock = $sock.getSocket();
```

Returns

the socket file descriptor number or -1 if the socket is not open

45.48.2.28 hash Qore::Socket::getSocketInfo (bool *host_lookup* = True)

Returns information about the local socket as a hash.

If the socket is not open, an exception is thrown

Example:

```
my hash $h = $sock.getSocketInfo();
```

Parameters

host_lookup	do a lookup of the hostname; if this is False then "hostname" and "hostname_desc" are not present in the response hash

Returns

a hash of information about the remote end for connected sockets

Exceptions

SOCKET-GETSOCKETINFO-ERROR	Socket is not open or error in getsockname()

Since

Qore 0.8.8 added the *host_lookup* parameter

45.48.2.29 __7_ string Qore::Socket::getSSLCipherName ( )

Returns the name of the cipher for an encrypted connection or NOTHING if a secure connection has not been established.

Code Flags:

CONSTANT

Example:

```
my *string $str = $sock.getSSLCipherName();
```

Returns

the name of the cipher for an encrypted connection or NOTHING if a secure connection has not been established

45.48.2.30 __7_ string Qore::Socket::getSSLCipherVersion ( )

Returns the version string of the cipher for an encrypted connection or NOTHING if a secure connection has not been established.

Code Flags:

CONSTANT

Example:

```
my *string $str = $sock.getSSLCipherVersion();
```

Returns

the version string of the cipher for an encrypted connection or NOTHING if a secure connection has not been established

45.48.2.31 hash Qore::Socket::getUsageInfo ( )

Returns performance statistics for the socket.

Code Flags:

CONSTANT

Example:

```
my hash $h = $sock.getUsageInfo();
```

Returns

a hash with the following keys:

- `"bytes_sent"`: an integer giving the total amount of bytes sent
- `"bytes_recv"`: an integer giving the total amount of bytes received
- `"us_sent"`: an integer giving the total number of microseconds spent sending data
- `"us_recv"`: an integer giving the total number of microseconds spent receiving data
- `"arg"`: (only if warning values have been set with Socket::setWarningQueue()) the optional argument for warning hashes
- `"timeout"`: (only if warning values have been set with Socket::setWarningQueue()) the warning timeout in microseconds
- `"min_throughput"`: (only if warning values have been set with Socket::setWarningQueue()) the minimum warning throughput in bytes/sec

Since

Qore 0.8.9

See also

Socket::clearStats()

45.48.2.32 bool Qore::Socket::isDataAvailable (timeout *timeout_ms* = 0)

Returns True or False depending on whether there is data to be read on the socket.

With a timeout of zero this method returns immediately and can be used for non-blocking polling the socket for data (can also be used to poll for new connections before Socket::accept()).

Example:

```
my bool $b = $sock.isDataAvailable();
```

Parameters

timeout_ms	an optional timeout in milliseconds (1/1000 second); if no timeout is given, the method returns immediately; Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

True if data is available on the Socket within the timeout period, False if not

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end closed the connection without sending any data

45.48.2.33 bool Qore::Socket::isOpen ( )

Returns True if the socket is open.

Code Flags:

CONSTANT

Example:

```
my bool $b = $sock.isOpen();
```

Returns

True if the socket is open, False if not

45.48.2.34 bool Qore::Socket::isSecure ( )

Returns True if the connection is a secure TLS/SSL connection.

Code Flags:

CONSTANT

Example:

```
my bool $b = $sock.isSecure();
```

Returns

True if the connection is encrypted, False if not

45.48.2.35 bool Qore::Socket::isWriteFinished (timeout *timeout_ms* = 0)

Returns True or False depending on whether all the data has been written to the socket.

With a timeout of zero this method returns immediately.

Example:

```
my bool $b = $sock.isWriteFinished();
```

Parameters

timeout_ms	an optional timeout in milliseconds (1/1000 second); if no timeout is given, the method returns immediately; Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

True if the send action completes within the timeout period, False if not

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end closed the connection without sending any data

45.48.2.36 int Qore::Socket::listen (int *backlog* = 20)

Listens for connections on a bound socket; sets the socket in a listening state.

Listens for new connections on a bound socket.

Example:

```
$sock.listen();
```

Parameters

backlog	the size of the queue for pending connections

Returns

Returns 0 for success, -1 for error

Exceptions

SOCKET-NOT-OPEN	The socket is not bound

Since

Qore 0.8.8 the *backlog* parameter was added

45.48.2.37 bool Qore::Socket::pendingHttpChunkedBody ( )

returns True if the socket is still connected, and a HTTP header was read indicating chunked transfer encoding, but no chunked body has been read yet

Example:

```
my bool $b = $sock.pendingHttpChunkedBody();
```

Since

Qore 0.8.10

45.48.2.38 hash Qore::Socket::readHTTPChunkedBody (timeout *timeout_ms* = -1)

Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the "body" key of a hash (including footers received)

If any errors are encountered, an exception is raised

Example:

```
my hash $ans = $sock.readHTTPChunkedBody(20s);
```

Events:

EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED

Parameters

timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a "SOCKET-TIMEOUT" exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

a hash of any HTTP footers received (with footer keys converted to lower case), with the body returned in the "body" key as a string in the Socket's encoding

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end has closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
READ-HTTP-CHUNK-ERROR	negative value given for chunk size by server

45.48.2.39 hash Qore::Socket::readHTTPChunkedBodyBinary (timeout *timeout_ms* = -1)

Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a binary object in the "body" key of a hash (including footers received)

If any errors are encountered, an exception is raised

Example:

```
my hash $ans = $sock.readHTTPChunkedBodyBinary(20s);
```

Events:

EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED

Parameters

timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a "SOCKET-TIMEOUT" exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

a hash of any HTTP footers received (with footer keys converted to lower case), with the body returned in the "body" key as a binary object

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end has closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
READ-HTTP-CHUNK-ERROR	negative value given for chunk size by server

45.48.2.40 Qore::Socket::readHTTPChunkedBodyBinaryWithCallback (code *rcb*, timeout *timeout_ms* = -1)

Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the "body" key of a hash (including footers received)

If any errors are encountered, an exception is raised

Example:

```
my hash $ans = $sock.readHTTPChunkedBodyBinaryWithCallback($recv_callback, 20s);
```

Events:

EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED

Parameters

rcb	The receive callback for the data received; first this method is called with a hash of the message headers, and then with any message body; if a chunked HTTP message is received, then the callback is called once for each chunk; when the message has been received, then the receive callback is called with a hash representing any trailer data received in a chunked transfer or NOTHING if the data was received in a normal message body or if there was no trailer data in a chunked transfer. The argument to this callback is always a hash; data calls have the following keys: • `"data"`: the string or binary data • `"chunked"`: True if the data was received with chunked transfer encoding, False if not Header or trailer data is placed in a hash with the following keys: • `"hdr"`: this can be assigned to NOTHING for the trailer hash if the data was not sent chunked or no trailers were included in a chunked message • `"obj"`: this is the owning object (so socket parameters can be changed based on headers received, such as, for example, socket character encoding)
timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a `"SOCKET-TIMEOUT"` exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end has closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
READ-HTTP-CHUNK-ERROR	negative value given for chunk size by server

Since

Qore 0.8.10

45.48.2.41 Qore::Socket::readHTTPChunkedBodyWithCallback (code *rcb*, timeout *timeout_ms* = `-1`)

Reads in an HTTP message body sent in chunked transfer encoding and returns it with any footers received as a string in the `"body"` key of a hash (including footers received)

If any errors are encountered, an exception is raised

Example:

```
my hash $ans = $sock.readHTTPChunkedBodyWithCallback($recv_callback, 20s);
```

Events:

EVENT_HTTP_CHUNK_SIZE, EVENT_HTTP_FOOTERS_RECEIVED

Parameters

rcb	The receive callback for the data received; first this method is called with a hash of the message headers, and then with any message body; if a chunked HTTP message is received, then the callback is called once for each chunk; when the message has been received, then the receive callback is called with a hash representing any trailer data received in a chunked transfer or NOTHING if the data was received in a normal message body or if there was no trailer data in a chunked transfer. The argument to this callback is always a hash; data calls have the following keys: • `"data"`: the string or binary data • `"chunked"`: True if the data was received with chunked transfer encoding, False if not Header or trailer data is placed in a hash with the following keys: • `"hdr"`: this can be assigned to NOTHING for the trailer hash if the data was not sent chunked or no trailers were included in a chunked message • `"obj"`: this is the owning object (so socket parameters can be changed based on headers received, such as, for example, socket character encoding)
timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a `"SOCKET-TIMEOUT"` exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end has closed the connection
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
READ-HTTP-CHUNK-ERROR	negative value given for chunk size by server

Since

Qore 0.8.10

45.48.2.42 hash Qore::Socket::readHTTPHeader (timeout *timeout_ms* = `-1`, __7_ reference *info*)

Retuns a hash representing the data in the HTTP header read, or, if the data cannot be parsed as an HTTP header, then an exception is thrown, and the data read is returned as a string in the `arg` key of the exception hash.

If any errors occur reading from the socket or if invalid HTTP data is received, an exception is raised. Accepts an optional timeout value in milliseconds.

Example:

```
my hash $ans = $sock.readHTTPHeader(10s);
```

Events:

EVENT_PACKET_READ, EVENT_HTTP_MESSAGE_RECEIVED

Parameters

timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a "SOCKET-TIMEOUT" exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)
info	An optional reference to an lvalue that can accept a hash that will be used as an output variable with one of the following two keys: • request-uri: (only set when parsing a request header) gives the request URI in an HTTP request • response-uri: (only set when parsing a response header) gives the response URI in an HTTP response • body-content-type: this is the "Content-Type" header without any charset declaration • charset: if there is a charset declaration in the "Content-Type" header, the value is returned in this key • close: (only set when parsing a request header) set to True if the connection should be closed after responding, False if not; see notes below about how this value is calculated • accept-charset: this key will be set to an appropriate value from any "Accept-Charset" header; if any of "*", "utf8", or "utf-8" are present, then this will be set to "utf8", otherwise it will be set to the first requested character encoding in the list • accept-encoding: this key will be set to a list of values from any "Accept-Encoding" header

Returns

a hash of headers (where each header key is converted to lower-case) also including the following keys giving additional information about the HTTP header received:

- http_version: a string giving the HTTP version set in the header
- status_code: (HTTP responses only) an integer giving the status code
- status_message: (HTTP responses only) if present in an HTTP response, this key will be set to the message after the status code
- method: (HTTP requests only) a string giving the HTTP method (i.e. "GET", "POST", etc)
- path: (HTTP requests only) a string giving the path in a request without any decoding; use decode_url() to decode if necessary

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-CLOSED	The remote end closed the connection without sending any data
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket

SOCKET-HTTP-ERROR	Invalid HTTP data was received, in the case of invalid header info received, the `arg` key of the exception hash will have the invalid data received

Note

- if the header claims a certain character encoding via a `charset` declaration in the `"Content-Type"` header, the Socket's character encoding is automatically set accordingly
- `RFC 2616 3.7.1`: if no encoding is specified, then set `"iso-8859-1"`
- `RFC 2068 19.7.1`: Persistent connections in HTTP/1.0 must be explicitly negotiated as they are not the default behavior
- `RFC 1945 1.3`: Except for experimental applications, current practice requires that the connection be established by the client prior to each request and closed by the server after sending the response.
- the conclusion is that server applications MUST close the connection when a request is received by an HTTP 1.0 client without an explicit request to keep the connection open

Since

- Qore 0.8.4 this method always returns a hash and raises a `SOCKET-HTTP-ERROR` if invalid HTTP data is received
- Qore 0.8.8 added the `close`, `charset`, `body-content-type`, and `accept-charset` info keys as well as the encoding handling based on the detected charset

45.48.2.43 string Qore::Socket::readHTTPHeaderString (timeout *timeout_ms* = `-1`)

Retuns a string representing the data in the HTTP header read (reads until `"\r\n\r\n"`)

If any errors occur reading from the socket, an exception is raised. Accepts an optional timeout value in milliseconds.

Example:

```
my string $str = $sock.readHTTPHeaderString(10s);
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	The timeout in milliseconds (1/1000 second). If no timeout or if a negative timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. If a timeout occurs, a `"SOCKET-TIMEOUT"` exception is raised. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

a string representing the header data read

Exceptions

SOCKET-NOT-OPEN	The socket is not connected

SOCKET-CLOSED	The remote end closed the connection without sending any data
SOCKET-RECV-ERROR	There was an error receiving the data
SOCKET-TIMEOUT	The data requested was not received in the timeout period
SOCKET-SSL-ERROR	There was an SSL error while reading data from the socket
SOCKET-HTTP-ERROR	maximum header size was exceeded

Since

Qore 0.8.8

45.48.2.44 string Qore::Socket::recv (softint *size* = `0`, timeout *timeout_ms* = `-1`)

Receives data from the socket and returns a string tagged with the Socket's character encoding.

If any errors occur reading from the socket, an exception is raised

Example:

```
my string $data = $sock.recv(-1); # read all data available
```

Events:

EVENT_PACKET_READ

Parameters

size	the amount of data to read in bytes; to read until the remote closes the connection, use -1; a value of 0 means to read all data currently available or available up until the timeout period (if non-negative)
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

the data read, returned as a string tagged with the Socket's character encoding

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection.
SOCKET-RECV-ERROR	there was an error receiving the data.
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

- Socket::setEncoding()
- Socket::recvBinary()

45.48.2.45 binary Qore::Socket::recvBinary (softint *size* = `0`, timeout *timeout_ms* = `-1`)

Receives data from the socket and returns a binary object.

If any errors occur reading from the socket, an exception is raised

Example:

```
my binary $data = $sock.recvBinary(-1); # read all data available
```

Events:

EVENT_PACKET_READ

Parameters

size	the amount of data to read in bytes; to read until the remote closes the connection, use -1; a value of 0 means to read all data currently available or available up until the timeout period (if non-negative)
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

the data read, returned as a binary object

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection.
SOCKET-RECV-ERROR	there was an error receiving the data.
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recv()

45.48.2.46 int Qore::Socket::recvi1 (timeout *timeout_ms* = `-1`)

Receives a 1-byte signed integer from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi1();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 1-byte signed integer read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvu1()

45.48.2.47 int Qore::Socket::recvi2 (timeout *timeout_ms* = -1)

Receives a 2-byte (16-bit) signed integer in big-endian format (network byte order) from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi2();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 2-byte signed integer in big-endian format (network byte order) read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvu2(), Socket::recvi2LSB(), Socket::recvu2LSB()

45.48.2.48 int Qore::Socket::recvi2LSB (timeout *timeout_ms* = -1)

Receives a 2-byte (16-bit) signed integer in little-endian format from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi2LSB();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 2-byte signed integer in little-endian format read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi2(), Socket::recvu2(), Socket::recvu2LSB()

45.48.2.49 int Qore::Socket::recvi4 (timeout *timeout_ms* = `-1`)

Receives a 4-byte (32-bit) signed integer in big-endian format (network byte order) from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi4();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 4-byte signed integer in big-endian format (network byte order) read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvu4(), Socket::recvi4LSB(), Socket::recvu4LSB()

45.48.2.50 int Qore::Socket::recvi4LSB (timeout *timeout_ms* = -1)

Receives a 4-byte (32-bit) signed integer in little-endian format from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi4LSB();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 4-byte signed integer in little-endian format read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi4(), Socket::recvu4(), Socket::recvu4LSB()

45.48.2.51 int Qore::Socket::recvi8 (timeout *timeout_ms* = -1)

Receives an 8-byte (64-bit) signed integer in big-endian format (network byte order) from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi8();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 8-byte signed integer in big-endian format (network byte order) read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi8LSB()

45.48.2.52 int Qore::Socket::recvi8LSB (timeout *timeout_ms* = `-1`)

Receives an 8-byte (64-bit) signed integer in little-endian format from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi8LSB();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 8-byte signed integer in little-endian format read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi8()

45.48.2.53 int Qore::Socket::recvu1 (timeout *timeout_ms* = -1)

Receives a 1-byte unsigned integer from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvu1();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

The 1-byte unsigned integer read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi1()

45.48.2.54 int Qore::Socket::recvu2 (timeout *timeout_ms* = -1)

Receives a 2-byte (16-bit) unsigned integer in big-endian format (network byte order) from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvu2();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

The 2-byte unsigned integer in big-endian format (network byte order) read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi2(), Socket::recvi2LSB(), Socket::recvu2LSB()

45.48.2.55 int Qore::Socket::recvu2LSB (timeout *timeout_ms* = -1)

Receives a 2-byte (16-bit) unsigned integer in little-endian format from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvi2LSB();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 2-byte unsigned integer in little-endian format read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi2(), Socket::recvu2(), Socket::recvi2LSB()

45.48.2.56 int Qore::Socket::recvu4 (timeout *timeout_ms* = -1)

Receives a 4-byte (32-bit) unsigned integer in big-endian format (network byte order) from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvu4();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 4-byte unsigned integer in big-endian format (network byte order) read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi4(), Socket::recvi4LSB(), Socket::recvu4LSB()

45.48.2.57 int Qore::Socket::recvu4LSB (timeout *timeout_ms* = `-1`)

Receives a 4-byte (32-bit) unsigned integer in little-endian format from the socket.

If any errors occur reading from the socket, an exception is raised

Example:

```
my int $val = $sock.recvu4LSB();
```

Events:

EVENT_PACKET_READ

Parameters

timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been read or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

The 4-byte unsigned integer in little-endian format read; if any errors occur reading from the socket, an exception is raised

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-CLOSED	the remote end has closed the connection
SOCKET-RECV-ERROR	there was an error receiving the data
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SSL-ERROR	there was an SSL error while reading data from the socket

See also

Socket::recvi4(), Socket::recvu4(), Socket::recvi4LSB()

45.48.2.58 int Qore::Socket::send (binary *bin*, int *timeout_ms* = -1)

Sends binary data over the socket; if any errors occur, an exception is thrown.

Example:

```
$sock.send($data, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

bin	Sends the binary data over the socket
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

- Socket::sendBinary()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::send2() except for the return value

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error like Socket::send2(); this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.59 int Qore::Socket::send (string *str*, timeout *timeout_ms* = -1)

Sends string data over the socket; string data is converted to the socket's encoding if necessary; if any errors occur, an exception is thrown.

String data will be converted to the encoding set for the socket if necessary.

Example:

```
$sock.send($str, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

str	sends the string data over the socket without the trailing null ('\0') character; the string's encoding is converted to the socket's encoding if necessary
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

- Socket::sendBinary2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::send2() except for the return value

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error like Socket::send2(); this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.60 nothing Qore::Socket::send2 (**binary** *bin,* **timeout** *timeout_ms* **=** `-1`)

Sends binary data over the socket; if any errors occur, an exception is thrown.

Example:

```
$sock.send2($data);
```

Events:

EVENT_PACKET_SENT

Parameters

bin	Sends the binary data over the socket
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

- Socket::sendBinary2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::send() except for the lack of a return value (since errors cause exceptions to be thrown, no return value is necessary)

Since

Qore 0.8.4

45.48.2.61 nothing Qore::Socket::send2 (string *str*, timeout *timeout_ms* = `-1`)

Sends string data over the socket; string data is converted to the socket's encoding if necessary; if any errors occur, an exception is thrown.

String data will be converted to the encoding set for the socket if necessary.

Example:

```
$sock.send2($str);
```

Events:

EVENT_PACKET_SENT

Parameters

str	sends the string data over the socket without the trailing null ('\0') character; the string's encoding is converted to the socket's encoding if necessary
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

- Socket::sendBinary2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::send() except for the lack of a return value (since errors cause exceptions to be thrown, no return value is necessary)

Since

Qore 0.8.4

45.48.2.62 int Qore::Socket::sendBinary (string *str*, timeout *timeout_ms* = -1)

Sends string data over the socket without converting the string to the socket's encoding, but instead is sent exactly as-is; if any errors occur, an exception is thrown.

Example:

```
$sock.sendBinary($str);
```

Events:

EVENT_PACKET_SENT

Parameters

str	string data to be sent (without any conversion to the socket's encoding) over the socket without the trailing null ('\0') character
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

Socket::send2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::sendBinary2() except for the return value

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error like Socket::send2(); this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.63 int Qore::Socket::sendBinary (binary *bin*, timeout *timeout_ms* = -1)

Sends binary data over the socket; if any errors occur, an exception is thrown.

Example:

```
$sock.sendBinary2($data);
```

Events:

EVENT_PACKET_SENT

Parameters

bin	Sends the binary data over the socket
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

Socket::send2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::sendBinary2() except for the return value

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error like Socket::send2(); this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.64 nothing Qore::Socket::sendBinary2 (string *str*, timeout *timeout_ms* = -1)

Sends string data over the socket without converting the string to the socket's encoding, but instead is sent exactly as-is; if any errors occur, an exception is thrown.

Example:

```
$sock.sendBinary2($str);
```

Events:

EVENT_PACKET_SENT

Parameters

str	string data to be sent (without any conversion to the socket's encoding) over the socket without the trailing null ('\0') character
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

Socket::send2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::sendBinary() except for the lack of a return value (since errors cause exceptions to be thrown, no return value is necessary)

Since

Qore 0.8.4

45.48.2.65 nothing Qore::Socket::sendBinary2 (**binary** *bin*, **timeout** *timeout_ms* = `-1`)

Sends binary data over the socket; if any errors occur, an exception is thrown.

Example:

```
$sock.sendBinary2($data);
```

Events:

EVENT_PACKET_SENT

Parameters

bin	Sends the binary data over the socket
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

See also

Socket::send2()

Note

this method is as of Qore 0.8.6+ equivalent to Socket::sendBinary() except for the lack of a return value (since errors cause exceptions to be thrown, no return value is necessary)

Since

Qore 0.8.4

45.48.2.66 nothing Qore::Socket::sendHTTPMessage (string *method*, string *path*, string *http_version*, hash *headers*, __7_ string *body*, __7_ reference *info*, timeout *timeout_ms* = -1)

Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.

Creates a properly-formatted HTTP message and sends it over the Socket. To send a string without converting the encoding to the Socket's encoding, convert the string to a binary object using the binary() function and use the variant of this method that takes a binary object for the data argument.

Example:

```
$sock.sendHTTPMessage("POST", "/RPC2", "1.1", ("Content-Type" : "text/xml"), $xml);
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

method	the HTTP method name to send (i.e. POST, HEAD, etc)
path	the path component of the URL
http_version	the HTTP protocol version ("1.0" or "1.1")
headers	a hash of additional headers to send (key-value pairs)
body	if present (and does not have a length of zero), the body to be sent with the message (if present, the "Content-Length" header will be automatically set); string data will be automatically converted to the Socket's encoding if necessary
info	An optional reference to an lvalue that can accept a hash that will be used as an output variable with the following hash key: • request-uri: The request URI as sent in the output message
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
ENCODING-CONVERSION-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-SEND-ERROR	Send failed
SOCKET-SSL-ERROR	there was an error sending SSL data

See also

Socket::sendHTTPResponse(), Socket::readHTTPHeader()

45.48.2.67 nothing Qore::Socket::sendHTTPMessage (string *method*, string *path*, string *http_version*, hash *headers*, binary *body*, __7_ reference *info*, timeout *timeout_ms* = -1)

Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.

Creates a properly-formatted HTTP message and sends it over the Socket

Example:

```
$sock.sendHTTPMessage("POST", "/RPC2", "1.1", ("Content-Type" : "text/xml"), $xml);
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

method	the HTTP method name to send (i.e. POST, HEAD, etc)
path	the path component of the URL
http_version	the HTTP protocol version ("1.0" or "1.1")
headers	a hash of additional headers to send (key-value pairs)
body	if it does not have a length of zero, the body to be sent with the message (if present, the "Content-Length" header will be automatically set)
info	An optional reference to an lvalue that can accept a hash that will be used as an output variable with the following hash key: • request-uri: The request URI as sent in the output message
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SEND-ERROR	Send failed
SOCKET-SSL-ERROR	there was an error sending SSL data

See also

Socket::sendHTTPResponse(), Socket::readHTTPHeader()

45.48.2.68 nothing Qore::Socket::sendHTTPMessageWithCallback (code *scb*, string *method*, string *path*, string *http_version*, hash *headers*, __7_ reference *info*, timeout *timeout_ms* = -1)

Sends an HTTP message with a method and user-defined headers given as a hash and an optional message body.

Creates a properly-formatted HTTP message and sends it over the Socket

Example:

```
$sock.sendHTTPMessageWithCallback($send_callback, "POST", "/RPC2", "1.1", ("Content-Type" : "text/xml"));
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

scb	The callback giving the chunked HTTP data to send; this callback must return either a string or a binary value each time it is called to give the chunked data to send; when all data has been sent, then a hash of message trailers can be sent or simply NOTHING which will close the chunked message
method	the HTTP method name to send (i.e. POST, HEAD, etc)
path	the path component of the URL
http_version	the HTTP protocol version (`"1.0"` or `"1.1"`)
headers	a hash of additional headers to send (key-value pairs)
info	An optional reference to an lvalue that can accept a hash that will be used as an output variable with the following hash key: • `request-uri`: The request URI as sent in the output message
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SEND-ERROR	Send failed
SOCKET-SSL-ERROR	there was an error sending SSL data

See also

Socket::sendHTTPResponse(), Socket::readHTTPHeader()

Since

Qore 0.8.10

45.48.2.69 nothing Qore::Socket::sendHTTPResponse (softint *status_code*, string *status_desc*, string *http_version*, hash *headers*, *__7_* string *body*, timeout *timeout_ms* = `-1`)

Sends an HTTP response with user-defined headers given as a hash and an optional message body.

Creates a properly-formatted HTTP response message and sends it over the Socket (must already be connected). If any errors occur, an exception is raised. To send a string without converting the encoding to the Socket's encoding, convert the string to a binary object using the binary() function and use the variant of this method that takes a binary object for the data argument.

Example:

```
$sock.sendHTTPResponse(200, "OK", "1.1", ("Connection":"Keep-Alive"));
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

status_code	the HTTP status code to send (i.e. `200`, `404`, etc)
status_desc	the descriptive text for the status code.
http_version	the HTTP protocol version (normally `"1.0"` or `"1.1"`, however this method allows any string to be sent)
headers	a hash of additional headers to send (key-value pairs)
body	if present (and does not have a length of zero), the body to be sent with the message (if present, the `"Content-Length"` header will be automatically set); string data will be automatically converted to the Socket's encoding if necessary
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-SENDHTTPRE↩ SPONSE-STATUS-ERR↩ OR	raised if the status_code (first argument) is $<$ 100 or $>$ 599
ENCODING-CONVERSI↩ ON-ERROR	the given string could not be converted to the socket's character encoding
SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SEND-ERROR	send failed

45.48.2.70 nothing Qore::Socket::sendHTTPResponse (softint *status_code*, string *status_desc*, string *http_version*, hash *headers*, binary *body*, timeout *timeout_ms* = `-1`)

Sends an HTTP response with user-defined headers given as a hash and a message body as literal binary data.

Creates a properly-formatted HTTP response message and sends it over the Socket (must already be connected). If any errors occur, an exception is raised.

Example:

```
$sock.sendHTTPResponse(200, "OK", "1.1", ("Connection":"Keep-Alive"), $bin);
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

status_code	the HTTP status code to send (i.e. `200`, `404`, etc)
status_desc	the descriptive text for the status code.
http_version	the HTTP protocol version (normally `"1.0"` or `"1.1"`, however this method allows any string to be sent)
headers	a hash of additional headers to send (key-value pairs)
body	if the binary object given does not have a length of zero, the body to be sent with the message (if present, the `"Content-Length"` header will be automatically set)
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Exceptions

SOCKET-SENDHTTPRE↩ SPONSE-STATUS-ERR↩ OR	raised if the status_code (first argument) is < 100 or > 599
SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SEND-ERROR	send failed

45.48.2.71 nothing Qore::Socket::sendHTTPResponseWithCallback (code *scb*, softint *status_code*, string *status_desc*, string *http_version*, hash *headers*, timeout *timeout_ms* = −1)

Sends an HTTP response with user-defined headers given as a hash and a message body as literal binary data.

Creates a properly-formatted HTTP response message and sends it over the Socket (must already be connected). If any errors occur, an exception is raised.

Example:

```
$sock.sendHTTPResponseWithCallback($send_callback, 200, "OK", "1.1", ("Connection":"Keep-Alive"));
```

Events:

EVENT_PACKET_SENT, EVENT_HTTP_SEND_MESSAGE

Parameters

scb	The callback giving the chunked HTTP data to send; this callback must return either a string or a binary value each time it is called to give the chunked data to send; when all data has been sent, then a hash of message trailers can be sent or simply NOTHING which will close the chunked message
status_code	the HTTP status code to send (i.e. 200, 404, etc)
status_desc	the descriptive text for the status code.
http_version	the HTTP protocol version (normally "1.0" or "1.1", however this method allows any string to be sent)
headers	a hash of additional headers to send (key-value pairs)
timeout_ms	the timeout in milliseconds (1/1000 second); If no timeout is passed or is negative, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Exceptions

SOCKET-SENDHTTPRE↩ SPONSE-STATUS-ERR↩ OR	raised if the status_code (first argument) is < 100 or > 599
SOCKET-NOT-OPEN	the socket is not connected
SOCKET-TIMEOUT	the data requested was not received in the timeout period
SOCKET-SEND-ERROR	send failed

Since

Qore 0.8.10

45.48.2.72 int Qore::Socket::sendi1 (softint *i* = 0, timeout *timeout_ms* = −1)

Sends a 1-byte integer over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi1($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send; only the least-significant byte will be sent
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.73 int Qore::Socket::sendi2 (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends a 2-byte (16-bit) integer in big-endian format (network byte order) over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi2($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send; only the least-significant 2 bytes will be sent
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.74 int Qore::Socket::sendi2LSB (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends a 2-byte (16-bit) integer in little-endian format over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi2LSB($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send; only the least-significant 2 bytes will be sent
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.75 int Qore::Socket::sendi4 (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends a 4-byte (32-bit) integer in big-endian format (network byte order) over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi4($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send; only the least-significant 4 bytes will be sent
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.76 int Qore::Socket::sendi4LSB (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends a 4-byte (32-bit) integer in little-endian format over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi4LSB($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send; only the least-significant 4 bytes will be sent
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.77 int Qore::Socket::sendi8 (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends an 8-byte (64-bit) integer in big-endian format (network byte order) over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi8($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.78 int Qore::Socket::sendi8LSB (softint *i* = `0`, timeout *timeout_ms* = `-1`)

Sends an 8-byte (64-bit) integer in little-endian format over the socket.

If any errors occur, an exception is thrown

Example:

```
$sock.sendi8LSB($val, 20s);
```

Events:

EVENT_PACKET_SENT

Parameters

i	the integer to send
timeout_ms	the timeout in milliseconds (1/1000 second). If no timeout is passed, then the call will not time out and will not return until all the data has been sent or the remote end closes the connection; the timeout value is the longest value that a single send() operation can take with non-blocking I/O. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

always returns 0 for backwards compatibility; if an error occurs, an exception is thrown

Exceptions

SOCKET-NOT-OPEN	The socket is not connected
SOCKET-TIMEOUT	a single send() operation exceeded the given timeout period
SOCKET-SEND-ERROR	an error occurred sending the socket data
SOCKET-SSL-ERROR	there was an SSL error while writing data to the socket

Since

Qore 0.8.6 this method takes a timeout value and throws exceptions on error; this change was made because it was not possible to catch SSL errors with this method with the previous implementation

45.48.2.79 nothing Qore::Socket::setCertificate (**SSLCertificate** *cert*)

Sets the X.509 certificate to use for negotiating encrypted connections.

Example:

```
$sock.setCertificate($cert);
```

Parameters

cert	This must be an SSLCertificate object

45.48.2.80 nothing Qore::Socket::setCertificate (**string** *cert_pem*)

Sets the X.509 certificate to use for negotiating encrypted connections from the PEM-encoded string representing the X.509 certificate.

Example:

```
$sock.setCertificate($cert_string);
```

Parameters

cert_pem	the PEM-encoded string representing the X.509 certificate

45.48.2.81 nothing Qore::Socket::setCertificate (binary *cert_der*)

Sets the X.509 certificate to use for negotiating encrypted connections from the DER-encoded binary object representing the X.509 certificate.

Example:

```
$sock.setCertificate($cert_der);
```

Parameters

cert_der	the DER-encoded binary object representing the X.509 certificate

45.48.2.82 nothing Qore::Socket::setCharset (string *encoding*)

Sets the character encoding for the socket.

A method synonym for Socket::setEncoding(), kept for backwards-compatibility

Parameters

encoding	The character encoding for the socket

45.48.2.83 nothing Qore::Socket::setEncoding (string *encoding*)

Sets the character encoding for the socket.

Parameters

encoding	the character encoding for the socket

45.48.2.84 nothing Qore::Socket::setEventQueue ( )

Removes any Queue object from the Socket object so that socket events are no longer added to the Queue.

Example:

```
$sock.setEventQueue();
```

See also

I/O Event Handling for more information

45.48.2.85 nothing Qore::Socket::setEventQueue (Queue *queue*)

Sets a Queue object to receive socket events.

Example:

```
$sock.setEventQueue($queue);
```

Parameters

queue	the Queue object to receive socket events; note that the Queue passed cannot have any maximum size set or a QUEUE-ERROR will be thrown

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set

See also

I/O Event Handling for more information

45.48.2.86 int Qore::Socket::setNoDelay (bool *nd* = True)

Sets the boolean `TCP_NODELAY` setting for the socket.

When this setting is True, then data will be immediately sent out over the socket, when it is False, then data transmission may be delayed to be packaged with other data for the same target.

Delayed data transmissions may cause problems when the sender immediately closes the socket after sending data; in this case the receiver may not get the data even though the send succeeded.

Note that if no value is given to the method, the argument will be assumed to be True, and output buffering will be turned off for the socket.

Example:

```
if ($sock.setNoDelay(True))
    printf("error setting TCP_NODELAY: %s\n", strerror(errno()));
```

Parameters

nd	the boolean `TCP_NODELAY` setting for the socket

Returns

0 for success, non-zero for errors; to get error information, see errno() and strerror()

See also

Socket::getNoDelay()

45.48.2.87 nothing Qore::Socket::setPrivateKey (SSLPrivateKey *key*)

Sets the private key to use for negotiating encrypted connections along with the X.509 certificate.

Example:

```
$sock.setPrivateKey($key);
```

Parameters

key	the private key for the X.509 certificate to use when establishing TLS/SSL encrypted connections

45.48.2.88 nothing Qore::Socket::setPrivateKey (string *key_pem*, __7_ string *pass*)

Sets the private key to use for negotiating encrypted connections along with the X.509 certificate from a PEM-encoded string representing the private key and an optional password.

Example:

```
$sock.setPrivateKey($key_pem, $password);
```

Parameters

key_pem	the PEM-encoded string representing the private key for the X.509 certificate to use when establishing TLS/SSL encrypted connections
pass	the password for the private key, if needed

45.48.2.89 nothing Qore::Socket::setPrivateKey (binary *key_der*)

Sets the private key to use for negotiating encrypted connections along with the X.509 certificate from a DER-encoded binary object representing the private key.

Example:

```
$sock.setPrivateKey($key_der);
```

Parameters

key_der	the DER-encoded binary object representing the private key for the X.509 certificate to use when establishing TLS/SSL encrypted connections

45.48.2.90 int Qore::Socket::setRecvTimeout (timeout *timeout_ms*)

sets the receive timeout as a socket option

Returns 0 for success, -1 for error (check errno() for the actual error in this case)

Example:

```
if ($sock.setRecvTimeout(20s))
    printf("error setting timeout on socket: %s\n", strerror(errno()));
```

Parameters

timeout_ms	the receive timeout to set with a resolution of milliseconds; Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. `2m` = two minutes, etc.)

Returns

0 for success, -1 for error (check errno() for the actual error in this case)

45.48.2.91 int Qore::Socket::setSendTimeout (timeout *timeout_ms*)

sets the send timeout as a socket option

Returns 0 for success, -1 for error (check errno() for the actual error in this case)

Example:

```
if ($sock.setSendTimeout(20s))
    printf("error setting timeout on socket: %s\n", strerror(errno()));
```

Parameters

timeout_ms	the send timeout to set with a resolution of milliseconds

Returns

0 for success, -1 for error (check errno() for the actual error in this case)

45.48.2.92 nothing Qore::Socket::setWarningQueue (int *warning_ms*, int *warning_bs*, Queue *queue*, any *arg*, timeout *min_ms* = `1s`)

Sets a Queue object to receive socket warnings.

Example:

```
$sock.setWarningQueue(5000, 5000, $queue, "socket-1");
```

Parameters

warning_ms	the threshold in milliseconds for individual socket actions (send, receive, connect), if exceeded, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-OPERATION-WARNING"` • `"operation"`: a string giving the operation that caused the warning (ex↩ : `"connect"`) • `"us"`: an integer giving the number of microseconds for the operation • `"timeout"`: an integer giving the warning threshold in microseconds • `"arg"`: if any `"arg"` argument is passed to the Socket::setWarningQueue() method, it will be included in the warning hash here
warning_bs	value in bytes per second; if any call has performance below this threshold, a socket warning is placed on the warning queue with the following keys: • `"type"`: a string with the constant value `"SOCKET-THROUGHPUT-WARNING"` • `"dir"`: either `"send"` or `"recv"` depending on the direction of the data flow • `"bytes"`: the amount of bytes sent • `"us"`: an integer giving the number of microseconds for the operation • `"bytes_sec"`: a float giving the transfer speed in bytes per second • `"threshold"`: an integer giving the warning threshold in bytes per second • `"arg"`: if any `"arg"` argument is passed to the Socket::setWarningQueue() method, it will be included in the warning hash here

queue	the Queue object to receive warning events
arg	an optional argument to be placed in the "arg" key in each warning hash (could be used to identify the socket for example)
min_ms	the minimum transfer time with a resolution of milliseconds for a transfer to be eligible for triggering a warning; transfers that take less than this period of time are not eligible for raising a warning

Exceptions

QUEUE-ERROR	the Queue passed has a maximum size set
SOCKET-SETWARNING↩ *QUEUE-ERROR*	at least one of *warning_ms* and *warning_bs* must be > 0

See also

Socket::clearWarningQueue()

Since

Qore 0.8.9

45.48.2.93 int Qore::Socket::shutdown ( )

Ensures that a socket will be closed even if the file descriptor is shared with other processes (for example, after a call to fork())

Does not throw any exceptions; returns an error code in case of an error

Example:

```
if ($sock.shutdown())
    stderr.printf("Error calling Socket::shutdown(): %s\n", strerror(
      errno()));
```

Returns

0 for success, -1 for error; in this case check errno() for the error number

45.48.2.94 nothing Qore::Socket::shutdownSSL ( )

Shuts down the SSL connection on a secure connection.

If any errors occur, an exception is raised

Example:

```
$sock.shutdownSSL();
```

Exceptions

SOCKET-SSL-ERROR	an error occurred shutting down the TLS/SSL connection

45.48.2.95 nothing Qore::Socket::upgradeClientToSSL ( )

Upgrades a client socket connection to a TLS/SSL connection.

Example:

```
$sock.upgradeClientToSSL();
```

45.48.2.96 nothing Qore::Socket::upgradeServerToSSL ( )

Upgrades a server socket connection to a TLS/SSL connection.

Example:

```
$sock.upgradeServerToSSL();
```

45.48.2.97 __7_ string Qore::Socket::verifyPeerCertificate ( )

Returns a string code giving the result of verifying the remote certificate or NOTHING if an encrypted connection is not currently established.

Code Flags:

CONSTANT

Example:

```
my *string $str = $sock.verifyPeerCertificate();
```

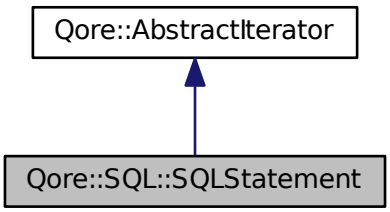
Returns

A string code giving the result of verifying the peer's certificate. No value is returned if a secure connection has not been established. The set of possible return values is made up of the keys of the X509_Verification↩ Reasons hash. This hash can also be used to generate a textual description of the verification result.

45.49 Qore::SQL::SQLStatement Class Reference

The SQLStatement class provides the most flexibilty for executing SQL on a database server.

Inheritance diagram for Qore::SQL::SQLStatement:



Public Member Functions

- bool active ()

  Returns True if the object is currently active and has a connection or transaction lock allocated to it, or False if not.
- int affectedRows ()

  Returns the number of rows affected by the last call to SQLStatement::exec()
- nothing beginTransaction ()

Manually starts a transaction and allocates a connection or grabs the transaction lock according to the object used in the SQLStatement::constructor()

- nothing bind (...)

  Binds placeholder buffer specifications and values to buffers defined in SQLStatement::prepare()
- nothing bindArgs (softlist vargs)

  Binds placeholder buffer specifications and values given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()
- nothing bindPlaceholders (...)

  Binds placeholder buffer specifications to buffers defined in SQLStatement::prepare()
- nothing bindPlaceholdersArgs (softlist vargs)

  Binds placeholder buffer specifications given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()
- nothing bindValues (...)

  Binds values to value buffer specifications to buffers defined in SQLStatement::prepare()
- nothing bindValuesArgs (softlist vargs)

  Binds values to value buffer specifications given as a list in the single argument to the method to value buffers defined in SQLStatement::prepare()
- nothing close ()

  Closes the statement if it is open, however this method does not release the connection or transaction lock.
- nothing commit ()

  Commits the transaction, releases the connection or the transaction lock according to the object used in the SQL↩ Statement::constructor(), and closes the SQLStatement.
- constructor (Datasource ds)

  Creates the SQLStatement object based on the given Datasource object that provides the connection to the database.
- constructor (DatasourcePool dsp)

  Creates the SQLStatement object based on the given DatasourcePool object that provides the connection to the database.
- copy ()

  Throws an exception; objects of this class cannot be copied.
- nothing define ()

  Performs an explicit define operation on the SQLStatement.
- hash describe ()

  Describes columns in the statement result.
- destructor ()

  Closes the statement if it is open and destroys the object.
- nothing exec (...)

  Executes the bound statement with any bound buffers, also optionally allows binding placeholder buffer specifications and values to buffers defined in SQLStatement::prepare() before executing the statement.
- nothing execArgs (softlist vargs)

  Executes the bound statement with any bound buffers, also optionally allows binding placeholder buffer specifications and values given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()
- hash fetchColumns (softint rows=-1)

  Retrieves a block of rows as a hash of lists with the maximum number of rows determined by the argument passed; automatically advances the row pointer; with this call it is not necessary to call SQLStatement::next().
- __7_ hash fetchRow ()

  Retrieves the current row as a hash where the keys are the column names and the values are the column values.
- list fetchRows (softint rows=-1)

  Retrieves a block of rows as a list of hashes with the maximum number of rows determined by the argument passed; automatically advances the row pointer; with this call it is not necessary to call SQLStatement::next()
- hash getOutput ()

  Retrieves output buffers as a hash; result sets will be returned as hashes of lists.
- hash getOutputRows ()

  Retrieves output buffers as a hash; result sets will be returned as lists of hashes.

- __7_ string getSQL ()

  Returns the current SQL string set with the call to SQLStatement::prepare() or SQLStatement::prepareRaw() or NOTHING if no SQL has been set.

- __7_ hash getValue ()

  Retrieves the current row as a hash where the keys are the column names and the values are the column values.

- any memberGate (string key)

  This method allows SQLStatement objects to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

- bool next ()

  Increments the row pointer when retrieving rows from a select statement; returns True if there is a row to retrieve, False if not.

- nothing prepare (string sql,...)

  Saves an SQL statement that will be prepared and executed later, along with optional arguments.

- nothing prepareRaw (string sql)

  Saves an SQL statement that will be prepared and executed later.

- nothing rollback ()

  Closes the SQLStatement, performs a transaction rollback, and releases the connection or the transaction lock according to the object used in the SQLStatement::constructor(), and closes the SQLStatement.

- bool valid ()

  returns True if the object is currently pointing at a valid element, False if not (use when iterating with SQLStatement::next())

45.49.1 Detailed Description

The SQLStatement class provides the most flexibilty for executing SQL on a database server.

Restrictions:

Qore::PO_NO_DATABASE

This class allows statements to be executed and result sets to be iteratively returned in all formats supported by Qore (single row at a time as a hash or blocks of rows as either hashes of lists or lists of hashes). The same flexibility of choosing the output format for result sets also applies to output values from stored procedure or function execution, for example.

This class does not differentiate between executing select statement or stored procedures or functions or other SQL code; the transaction lock for Datasource objects and a dedicated connection for DatasourcePool objects is allocated to the object when the connection is necessary, and it is not automatically released by this class. The transaction lock or connection must be manually released by calling SQLStatement::commit() or SQLStatement::rollback() (or the methods with the same name in the parent Datasource or DatasourcePool object).

Here is an example executing a select statement:

```
{
    my SQLStatement $stmt($ds);
    # release transaction lock on exit
    on_exit $stmt.commit();
    $stmt.prepare("select * from table");
    while ($stmt.next()) {
        my hash $row = $stmt.fetchRow();
        do_something($row);
    }
}
```

Note

- Most commands are executed implicitly; for example, in the example above there is no call to SQLStatement::exec() as it is executed implicitly in the initial call to SQLStatement::next().
- Current column values in query results iterated with SQLStatement::next() as above can also be dereferenced directly from the SQLStatement object by using the column name in lower case as a member name (using SQLStatement::memberGate(), see that method for an example)
- Query results can also be returned in blocks using SQLStatement::fetchRows() and SQLStatement::fetchColumns() ("rows" and "columns" in this case refer to the output data format; also using these methods there is no need to call SQLStatement::next())

The following methods are useful when executing all statements:

- SQLStatement::prepare()
- SQLStatement::prepareRaw()
- SQLStatement::bind()
- SQLStatement::bindArgs()
- SQLStatement::bindPlaceholders()
- SQLStatement::bindPlaceholdersArgs()
- SQLStatement::bindValues()
- SQLStatement::bindValuesArgs()
- SQLStatement::exec()
- SQLStatement::execArgs()
- SQLStatement::beginTransaction()
- SQLStatement::commit()
- SQLStatement::rollback()
- SQLStatement::close()
- SQLStatement::getSQL()
- SQLStatement::active()

The following methods are useful when executing select statements:

- SQLStatement::next()
- SQLStatement::fetchRow()
- SQLStatement::fetchRows()
- SQLStatement::fetchColumns()

The following methods are useful when executing stored procedures, functions, or other non-select SQL statements:

- SQLStatement::getOutput()
- SQLStatement::getOutputRows()

Note

This class is not available with the PO_NO_DATABASE parse option

45.49.2 Member Function Documentation

45.49.2.1 bool Qore::SQL::SQLStatement::active ( )

Returns True if the object is currently active and has a connection or transaction lock allocated to it, or False if not.

Returns

True if the object is currently active and has a connection or transaction lock allocated to it, or False if not

Code Flags:

CONSTANT

Example:

```
if ($stmt.active())
    $stmt.commit();
```

45.49.2.2 int Qore::SQL::SQLStatement::affectedRows ( )

Returns the number of rows affected by the last call to SQLStatement::exec()

Returns

the number of rows affected by the last call to SQLStatement::exec()

Example:

```
my int $rc = $stmt.affectedRows();
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepare↩ Raw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed; see the relevant DBI driver docs for more information

45.49.2.3 nothing Qore::SQL::SQLStatement::beginTransaction ( )

Manually starts a transaction and allocates a connection or grabs the transaction lock according to the object used in the SQLStatement::constructor()

Example:

```
$stmt.beginTransaction();
```

45.49.2.4 nothing Qore::SQL::SQLStatement::bind (...)

Binds placeholder buffer specifications and values to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

Any arguments previously bound will be released when this call is made.

Note

You can also bind directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Parameters

...	Arguments to placeholder specifications (if required by the underlying DBI driver) and bind by value arguments

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    $stmt.bind($h.id, $h.name);
    $stmt.exec();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

See also

SQLStatement::bindArgs(), SQLStatement::bindPlaceholders(), SQLStatement::bindPlaceholdersArgs(), SQLStatement::bindValues(), and SQLStatement::bindValuesArgs()

45.49.2.5 nothing Qore::SQL::SQLStatement::bindArgs (softlist *vargs*)

Binds placeholder buffer specifications and values given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

Any arguments previously bound will be released when this call is made.

Note

You can also bind directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Parameters

vargs	Arguments to placeholder specifications (if required by the underlying DBI driver) and bind by value arguments

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    my list $args = ($h.id, $h.name);
    $stmt.bindArgs($args);
    $stmt.exec();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

See also

SQLStatement::bind(), SQLStatement::bindPlaceholders(), SQLStatement::bindPlaceholdersArgs(), SQLStatement::bindValues(), and SQLStatement::bindValuesArgs()

45.49.2.6 nothing Qore::SQL::SQLStatement::bindPlaceholders (...)

Binds placeholder buffer specifications to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method. Only placeholder buffer specifications will be processed; value buffer specifications will be skipped by this method.

Any buffer specifications previously defined will be released when this call is made.

Note

You can also bind buffer specifications directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Not all DBI drivers require binding placeholders specification.

Parameters

...	Arguments to placeholder specifications (if required by the underlying DBI driver)

Example:

```
$stmt.prepare("begin select sysdate into :sd from dual", Type::Date); end;
$stmt.bindPlaceholders(Type::Date);
my date $d = $stmt.getOutput().sd;
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

See also

SQLStatement::bind(), SQLStatement::bindArgs(), SQLStatement::bindPlaceholdersArgs(), SQLStatement::bindValues(), and SQLStatement::bindValuesArgs()

45.49.2.7 nothing Qore::SQL::SQLStatement::bindPlaceholdersArgs (softlist *vargs*)

Binds placeholder buffer specifications given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method. Only placeholder buffer specifications will be processed; value buffer specifications will be skipped by this method.

Any buffer specifications previously defined will be released when this call is made.

Note

You can also bind buffer specifications directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Not all DBI drivers require binding placeholders specification.

Parameters

vargs	Arguments to placeholder specifications (if required by the underlying DBI driver)

Example:

```
$stmt.prepare("begin select sysdate into :sd from dual", Type::Date); end;
my list $l = list(Type::Date);
$stmt.bindPlaceholdersArgs($l);
my date $d = $stmt.getOutput().sd;
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

See also

SQLStatement::bind(), SQLStatement::bindArgs(), SQLStatement::bindPlaceholders(), SQLStatement::bindValues(), and SQLStatement::bindValuesArgs()

45.49.2.8 nothing Qore::SQL::SQLStatement::bindValues (...)

Binds values to value buffer specifications to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

Any values previously bound will be released when this call is made.

Note

You can also bind directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Parameters

...	Arguments to bind by value arguments

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    $stmt.bindValues($h.id, $h.name);
    $stmt.exec();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

See also

SQLStatement::bind(), SQLStatement::bindArgs(), SQLStatement::bindPlaceholders(), SQLStatement::bindPlaceholdersArgs(), and SQLStatement::bindValuesArgs().

45.49.2.9 nothing Qore::SQL::SQLStatement::bindValuesArgs (softlist *vargs*)

Binds values to value buffer specifications given as a list in the single argument to the method to value buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

Any values previously bound will be released when this call is made.

Note

You can also bind directly when calling SQLStatement::exec() or SQLStatement::execArgs() as a shortcut as well, in which case it's not necessary to make an extra call to this method.

Parameters

vargs	Arguments to bind by value arguments

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    my list $args = ($h.id, $h.name);
    $stmt.bindValuesArgs($args);
    $stmt.exec();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications; see the relevant DBI driver docs for more information

45.49.2.10 nothing Qore::SQL::SQLStatement::close ( )

Closes the statement if it is open, however this method does not release the connection or transaction lock.

Example:

```
$stmt.close();
```

45.49.2.11 nothing Qore::SQL::SQLStatement::commit ( )

Commits the transaction, releases the connection or the transaction lock according to the object used in the SQL↩ Statement::constructor(), and closes the SQLStatement.

Example:

```
$stmt.commit();
```

Note

For possible exceptions; see DBI driver docs for the commit() method

45.49.2.12 Qore::SQL::SQLStatement::constructor (Datasource *ds*)

Creates the SQLStatement object based on the given Datasource object that provides the connection to the database.

This method will throw an exception only if the object passed as an argument uses a driver that does not support the prepared statement interface added in Qore version 0.8.1.

Parameters

ds	The Datasource object to use for the DB connection for the SQLStatement object

Example:

```
my SQLStatement $stmt($db);
```

Exceptions

SQLSTATEMENT-ERROR	the DBI driver for the given object does not support the prepared statement API

Since

Qore 0.8.1

45.49.2.13 Qore::SQL::SQLStatement::constructor (**DatasourcePool** *dsp*)

Creates the SQLStatement object based on the given DatasourcePool object that provides the connection to the database.

This method will throw an exception only if the object passed as an argument uses a driver that does not support the prepared statement interface added in Qore version 0.8.1.

Parameters

dsp	The DatasourcePool object to use for the DB connection for the SQLStatement object

Example:

```
my SQLStatement $stmt($db);
```

Exceptions

SQLSTATEMENT-ERROR	the DBI driver for the given object does not support the prepared statement API

Since

Qore 0.8.1

45.49.2.14 Qore::SQL::SQLStatement::copy ( )

Throws an exception; objects of this class cannot be copied.

Exceptions

SQLSTATEMENT-COPY-ERROR	SQLStatement objects cannot be copied

45.49.2.15 nothing Qore::SQL::SQLStatement::define ( )

Performs an explicit define operation on the SQLStatement.

It is not encessary to call this method manually; define operations are implicitly executed when needed when retrieving values from a select statement

Example:

```
{
    my SQLStatement $stmt($ds);
```

```
        # release transaction lock on exit
        on_exit $stmt.commit();
        $stmt.prepare("select * from table");
        $stmt.exec();
        $stmt.define();
        # note that the SQLStatement::next() would implicitly execute exec() and define()
        while ($stmt.next()) {
            my hash $row = $stmt.fetchRow();
            do_something($row);
        }
    }
```

45.49.2.16 hash Qore::SQL::SQLStatement::describe ( )

Describes columns in the statement result.

Return values

hash	with (column name : description hash) pairs

This method has to be called after first `next()` (or its equivalent) call.

Description Hash Structure

Key	Description
name	Column name. String.
type	Qore data type constant as used in Qore::zzz8valuezzz9::typeCode(). Integer.
maxsize	Maximum size of the data value as in the DB server. Integer.
native_type	Database data type. String.
internal_id	Database data type identifier. Integer.

45.49.2.17 Qore::SQL::SQLStatement::destructor ( )

Closes the statement if it is open and destroys the object.

Example:

```
    delete $stmt;
```

45.49.2.18 nothing Qore::SQL::SQLStatement::exec (...)

Executes the bound statement with any bound buffers, also optionally allows binding placeholder buffer specifications and values to buffers defined in SQLStatement::prepare() before executing the statement.

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Optional arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

If bind arguments are provided, any arguments previously bound will be released when this call is made.

After calling this method to execute the statement, to retrieve information about the call or output values bound in the call, call SQLStatement::affectedRows(), SQLStatement::getOutput(), or SQLStatement::getOutputRows() as needed.

To retrieve rows from a select statement call either SQLStatement::next() and SQLStatement::fetchRow(), or SQ↩LStatement::fetchRows() or SQLStatement::fetchColumns() as needed.

Parameters

...	Optional arguments to placeholder specifications (if required by the underlying DBI driver) and bind by value arguments can be given in the call to the method; if present, arguments are bound before the statement is executed

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    $stmt.exec($h.id, $h.name);
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed; see the relevant DBI driver docs for more information

See also

SQLStatement::execArgs()

45.49.2.19 nothing Qore::SQL::SQLStatement::execArgs (softlist *vargs*)

Executes the bound statement with any bound buffers, also optionally allows binding placeholder buffer specifications and values given as a list in the single argument to the method to buffers defined in SQLStatement::prepare()

If the statement has not previously been prepared with the DB API, it will be implicitly prepared by this method call. This means that this call will cause a connection to be dedicated from a DatasourcePool object or the transaction lock to be grabbed with a Datasource object, depending on the argument to SQLStatement::constructor().

Optional arguments to buffer specifications must be given in the same order as declared in the string given to the SQLStatement::prepare() method.

If bind arguments are provided, any arguments previously bound will be released when this call is made.

After calling this method to execute the statement, to retrieve information about the call or output values bound in the call, call SQLStatement::affectedRows(), SQLStatement::getOutput(), or SQLStatement::getOutputRows() as needed.

To retrieve rows from a select statement call either SQLStatement::next() and SQLStatement::fetchRow(), or SQ↩LStatement::fetchRows() or SQLStatement::fetchColumns() as needed.

Parameters

vargs	Optional arguments to placeholder specifications (if required by the underlying DBI driver) and bind by value arguments can be given in the call to the method; if present, arguments are bound before the statement is executed

Example:

```
$stmt.prepare("insert into table (id, name) values (%v, %v)");
foreach my hash $h in ($l) {
    my list $args = ($h.id, $h.name);
    $stmt.execArgs($args);
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed; see the relevant DBI driver docs for more information

See also

SQLStatement::exec()

45.49.2.20 hash Qore::SQL::SQLStatement::fetchColumns (softint *rows* = -1)

Retrieves a block of rows as a hash of lists with the maximum number of rows determined by the argument passed; automatically advances the row pointer; with this call it is not necessary to call SQLStatement::next().

If the argument passed is omitted or less than or equal to zero, then all available rows from the current row position are retrieved, also if fewer rows are available than requested then only the rows available are retrieved.

If no more rows are available then a hash is returned where each key value is an empty list.

Parameters

rows	The maximum number of rows to retrieve, if this argument is omitted, negative, or equal to zero, then all available rows from the current row position are retrieved

Example:

```
my hash $h = $stmt.fetchColumns(-1);
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

- There is no need to call SQLStatement::next() when calling this method; the method automatically iterates through the given number of rows
- Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when row values are retrieved; see the relevant DBI driver docs for more information

45.49.2.21 __7_ hash Qore::SQL::SQLStatement::fetchRow ( )

Retrieves the current row as a hash where the keys are the column names and the values are the column values.

Use with SQLStatement::next() to iterate through the results of a select statement one row at a time

Returns

the current row as a hash where the keys are the column names and the values are the column values

Example:

```
while ($stmt.next()) {
    my hash $h = $stmt.fetchRow();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when row values are retrieved; see the relevant DBI driver docs for more information

45.49.2.22 list Qore::SQL::SQLStatement::fetchRows (softint *rows* = -1)

Retrieves a block of rows as a list of hashes with the maximum number of rows determined by the argument passed; automatically advances the row pointer; with this call it is not necessary to call SQLStatement::next()

If the argument passed is omitted or less than or equal to zero, then all available rows from the current row position are retrieved, also if fewer rows are available than requested then only the rows available are retrieved.

If no more rows are available then an empty list is returned.

Parameters

rows	The maximum number of rows to retrieve, if this argument is omitted, negative, or equal to zero, then all available rows from the current row position are retrieved

Example:

```
my list $l = $stmt.fetchRows(-1);
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

- There is no need to call SQLStatement::next() when calling this method; the method automatically iterates through the given number of rows
- Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when row values are retrieved; see the relevant DBI driver docs for more information

45.49.2.23 hash Qore::SQL::SQLStatement::getOutput ( )

Retrieves output buffers as a hash; result sets will be returned as hashes of lists.

Returns

Returns a hash of output buffers; result sets will be returned as hashes of lists. Each key in the hash is the same as the name given to the placeholder specification in the call to SQLStatement::prepare() or SQLStatement::prepareRaw()

Example:

```
my hash $h = $stmt.getOutput();
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when output values are retrieved; see the relevant DBI driver docs for more information

45.49.2.24 hash Qore::SQL::SQLStatement::getOutputRows ( )

Retrieves output buffers as a hash; result sets will be returned as lists of hashes.

Returns

Retrieves output buffers as a hash; result sets will be returned as lists of hashes. Each key in the hash is the same as the name given to the placeholder specification in the call to SQLStatement::prepare() or SQLStatement::prepareRaw()

Example:

```
my hash $h = $stmt.getOutputRows();
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when output values are retrieved; see the relevant DBI driver docs for more information

45.49.2.25 __7_ string Qore::SQL::SQLStatement::getSQL ( )

Returns the current SQL string set with the call to SQLStatement::prepare() or SQLStatement::prepareRaw() or NOTHING if no SQL has been set.

Returns

Returns the current SQL string set with the call to SQLStatement::prepare() or SQLStatement::prepareRaw() or NOTHING if no SQL has been set

Example:

```
my *string $sql = $stmt.getSQL();
```

45.49.2.26 __7_ hash Qore::SQL::SQLStatement::getValue ( ) [virtual]

Retrieves the current row as a hash where the keys are the column names and the values are the column values.

Use with SQLStatement::next() to iterate through the results of a select statement one row at a time

Returns

the current row as a hash where the keys are the column names and the values are the column values

Example:

```
while ($stmt.next()) {
    my hash $h = $stmt.getValue();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

- Equivalent to SQLStatement::fetchRow()
- Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed or when row values are retrieved; see the relevant DBI driver docs for more information

Since

Qore 0.8.5

Implements Qore::AbstractIterator.

45.49.2.27 any Qore::SQL::SQLStatement::memberGate (string *key*)

This method allows SQLStatement objects to be dereferenced directly as a hash for the current row being iterated, as memberGate methods are called implicitly when an unknown member is accessed from outside the class.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the column name for the value to retrieve

Returns

the current column value of the given row

Example:

```
my SQLStatement $stmt($ds);
# release transaction lock on exit
on_exit $stmt.commit();
$stmt.prepare("select name, birthdate from table");
while ($stmt.next()) {
    printf("name: %s birthdate: %y", $stmt.name, $stmt.birthdate);
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()
SQLSTATEMENT-ITERATION-ERROR	Cannot dereference statement iteration context; make sure and call SQLStatement::next() before trying to dereference the current row being iterated
ENCODING-CONVERSION-ERROR	this error is thrown if the given key cannot be converted to the default character encoding
SQLSTATEMENT-COLUMN-ERROR	The given column name does not exist in the current row data

Note

Column values can only be dereferenced using the automatic SQLStatement::memberGate() method while iterating a result set with SQLStatement::next(); the hash that will be dereferenced is equivalent to that returned by SQLStatement::fetchRow()

Since

Qore 0.8.6

45.49.2.28 bool Qore::SQL::SQLStatement::next ( ) [virtual]

Increments the row pointer when retrieving rows from a select statement; returns True if there is a row to retrieve, False if not.

If this method returns True, then call SQLStatement::fetchRow() afterwards to retrieve the row

Returns

True if there is a row to retrieve, False if not (no more rows to be retrieved)

Example:

```
while ($stmt.next()) {
    my hash $h = $stmt.fetchRow();
}
```

Exceptions

SQLSTATEMENT-ERROR	No SQL has been set with SQLStatement::prepare() or SQLStatement::prepareRaw()

Note

Exceptions could be thrown by the DBI driver when the statement is prepared or when attempting to bind the given arguments to buffer specifications or when the statement is executed; see the relevant DBI driver docs for more information

Implements Qore::AbstractIterator.

45.49.2.29 nothing Qore::SQL::SQLStatement::prepare (string *sql*, ...)

Saves an SQL statement that will be prepared and executed later, along with optional arguments.

The statement is actually only prepared when used for the first time, this is so that SQLStatement objects created with DatasourcePool objects use the DatasourcePool more efficiently, as many drivers require the actual DB API prepare call to be made on the same connection as the connection the statement will be executed on as well.

Note

This method parses the SQL string for placeholders and bind by value tokens (`v`); for a version of this method that does not parse the SQL string for placeholders and bind by value tokens, see SQLStatement::prepareRaw().

Parameters

sql	The SQL string to prepare for execution on the DB server

Example:

```
$stmt.prepare("select * from table where id = %v");
```

45.49.2.30 nothing Qore::SQL::SQLStatement::prepareRaw (string *sql*)

Saves an SQL statement that will be prepared and executed later.

The statement is actually only prepared when used for the first time, this is so that SQLStatement objects created with DatasourcePool objects use the DatasourcePool more efficiently, as many drivers require the actual DB API prepare call to be made on the same connection as the connection the statement will be executed on as well.

Note

This method does not parse the SQL string for placeholders and bind by value tokens (%v); for a version of this method that does parse the SQL string for placeholders and bind by value tokens, see SQLStatement::prepare().

Parameters

sql	The SQL string to prepare for execution on the DB server

Example:

```
$stmt.prepareRaw("select * from table");
```

45.49.2.31 nothing Qore::SQL::SQLStatement::rollback ( )

Closes the SQLStatement, performs a transaction rollback, and releases the connection or the transaction lock according to the object used in the SQLStatement::constructor(), and closes the SQLStatement.

Example:

```
$stmt.rollback();
```

Note

For possible exceptions; see DBI driver docs for the rollback() method

45.49.2.32 bool Qore::SQL::SQLStatement::valid ( ) [virtual]

returns True if the object is currently pointing at a valid element, False if not (use when iterating with SQLStatement::next())

Returns

True if the object is currently pointing at a valid element, False if not

Code Flags:

CONSTANT

Example:

```
if ($i.valid())
    printf("current value: %y\n", $i.getValue());
```

Implements Qore::AbstractIterator.

45.50 Qore::SSLCertificate Class Reference

SSLCertificate objects allow Qore code to work with X.509 certificate data.

Public Member Functions

- constructor (string pem)

  Creates the SSLCertificate object from the PEM-encoded version of the X.509 certificate.
- constructor (binary der)

  Creates the SSLCertificate object from the DER-encoded version of the X.509 certificate.
- copy ()

  Copying objects of this class is not supported, an exception will be thrown.
- hash getInfo ()

  Returns a hash of all information for the certificate.
- hash getIssuerHash ()

  Returns a hash of strings representing the issuer information of the certificate.
- date getNotAfterDate ()

  Returns a date/time value representing the end date of the certificate.
- date getNotBeforeDate ()

  Returns a date/time value representing the start date of the certificate.
- string getPEM ()

  Returns a string in PEM format representing the certificate.
- __7_ binary getPublicKey ()

  Returns a binary object representing the public key of the certificate in DER (Distinguished Encoding Rules) format or NOTHING if no public key is present in the certificate.
- string getPublicKeyAlgorithm ()

  Returns the name of the public key algorithm of the certificate.
- hash getPurposeHash ()

  Returns a hash of booleans representing the allowed purposes of the certificate.
- int getSerialNumber ()

  Returns the integer serial number of the certificate.
- binary getSignature ()

  Returns a binary object representing the signature of the certificate.
- string getSignatureType ()

  Returns the signature type of the certificate.
- hash getSubjectHash ()

  Returns a hash of strings representing the subject information of the certificate.
- int getVersion ()

  Returns the version of the certificate as an integer.

45.50.1 Detailed Description

SSLCertificate objects allow Qore code to work with X.509 certificate data.

45.50.2 Member Function Documentation

45.50.2.1 Qore::SSLCertificate::constructor (string *pem*)

Creates the SSLCertificate object from the PEM-encoded version of the X.509 certificate.

Parameters

pem	the PEM representation of the X.509 certificate

Example:

```
my SSLCertificate $cert($pem_cert_string);
```

Exceptions

SSLCERTIFICATE-CONSTRUCTOR-ERROR	invalid X.509 certificate data

Since

0.8.4: the deprecated functionality where the if the string passed was less than 200 bytes long, it was assumed to be a file name has been removed; the string is assumed to be the PEM-encoded X.509 Certificate itself

45.50.2.2 Qore::SSLCertificate::constructor (**binary** *der*)

Creates the SSLCertificate object from the DER-encoded version of the X.509 certificate.

Parameters

der	the DER-encoded representation of the X.509 certificate

Example:

```
my SSLCertificate $cert($der_cert);
```

Exceptions

SSLCERTIFICATE-CONSTRUCTOR-ERROR	invalid X.509 certificate data

45.50.2.3 Qore::SSLCertificate::copy ( )

Copying objects of this class is not supported, an exception will be thrown.

Exceptions

SSLCERTIFICATE-COPY-ERROR	SSLCertificate objects cannot be copied

45.50.2.4 **hash** Qore::SSLCertificate::getInfo ( )

Returns a hash of all information for the certificate.

Returns

a hash of all information for the certificate with the following keys:

- `"version"`: The version of the X.509 certificate (see SSLCertificate::getVersion())
- `"serialNumber"`: The serial number of the X.509 certificate (see SSLCertificate::getSerialNumber())
- `"subject"`: The subject hash of the X.509 certificate (see SSLCertificate::getSubjectHash())
- `"issuer"`: The issuer hash of the X.509 certificate (see SSLCertificate::getIssuerHash())

- "purposes": The purpose hash of the X.509 certificate (see SSLCertificate::getPurposeHash())
- "notBefore": The "not before date" of the X.509 certificate (see SSLCertificate::getNotBeforeDate())
- "notAfter": The "not after date" of the X.509 certificate (see SSLCertificate::getNotAfterDate())
- "signatureType": The signature type string of the X.509 certificate (see SSLCertificate::getSignatureType())
- "signature": The binary signature of the X.509 certificate (see SSLCertificate::getSignature())
- "publicKey": The binary public key of the X.509 certificate (see SSLCertificate::getPublicKey())

Code Flags:

CONSTANT

Example:

```
my hash $hash = $cert.getInfo();
```

45.50.2.5 hash Qore::SSLCertificate::getIssuerHash ( )

Returns a hash of strings representing the issuer information of the certificate.

Returns

a hash of key-value pairs representing the issuer information of the certificate

Code Flags:

CONSTANT

Example:

```
my hash $h = $cert.getIssuerHash();
```

45.50.2.6 date Qore::SSLCertificate::getNotAfterDate ( )

Returns a date/time value representing the end date of the certificate.

Returns

a date/time value representing the end date of the certificate

Code Flags:

CONSTANT

Example:

```
my date $end = $cert.getNotAfterDate();
```

45.50.2.7 date Qore::SSLCertificate::getNotBeforeDate ( )

Returns a date/time value representing the start date of the certificate.

Returns

a date/time value representing the start date of the certificate

Code Flags:

CONSTANT

Example:

```
my date $start = $cert.getNotBeforeDate();
```

45.50.2.8 string Qore::SSLCertificate::getPEM ( )

Returns a string in PEM format representing the certificate.

Returns

a string in PEM format representing the certificate

Example:

```
my string $pem_str = $cert.getPEM();
```

Exceptions

X509-ERROR	could not create PEM string from X509 certificate data

45.50.2.9 __7_ binary Qore::SSLCertificate::getPublicKey ( )

Returns a binary object representing the public key of the certificate in DER (Distinguished Encoding Rules) format or NOTHING if no public key is present in the certificate.

Returns

a binary object representing the public key of the certificate in DER (Distinguished Encoding Rules) format or NOTHING if no public key is present in the certificate

Code Flags:

CONSTANT

Example:

```
my *binary $bin = $cert.getPublicKey();
```

45.50.2.10 string Qore::SSLCertificate::getPublicKeyAlgorithm ( )

Returns the name of the public key algorithm of the certificate.

Returns

the name of the public key algorithm of the certificate

Code Flags:

CONSTANT

Example:

```
my string $str = $cert.getPublicKeyAlgorithm();
```

45.50.2.11 hash Qore::SSLCertificate::getPurposeHash ( )

Returns a hash of booleans representing the allowed purposes of the certificate.

Returns

a hash of booleans representing the allowed purposes of the certificate

Code Flags:

CONSTANT

Example:

```
my hash $h = $cert.getPurposeHash();
```

45.50.2.12 int Qore::SSLCertificate::getSerialNumber ( )

Returns the integer serial number of the certificate.

Returns

the integer serial number of the certificate

Code Flags:

CONSTANT

Example:

```
my int $sn = $cert.getSerialNumber();
```

45.50.2.13 binary Qore::SSLCertificate::getSignature ( )

Returns a binary object representing the signature of the certificate.

Returns

a binary object representing the signature of the certificate

Code Flags:

CONSTANT

Example:

```
my binary $bin = $cert.getSignature();
```

45.50.2.14 string Qore::SSLCertificate::getSignatureType ( )

Returns the signature type of the certificate.

Returns

the signature type of the certificate

Code Flags:

CONSTANT

Example:

```
my string $str = $cert.getSignatureType();
```

45.50.2.15 hash Qore::SSLCertificate::getSubjectHash ( )

Returns a hash of strings representing the subject information of the certificate.

Returns

a hash of key-value pairs representing the subject information of the certificate

Code Flags:

CONSTANT

Example:

```
my hash $h = $cert.getSubjectHash();
```

45.50.2.16 int Qore::SSLCertificate::getVersion ( )

Returns the version of the certificate as an integer.

Returns

the version of the certificate as an integer

Code Flags:

CONSTANT

Example:

```
my int $ver = $cert.getVersion();
```

45.51 Qore::SSLPrivateKey Class Reference

This class implements a container for private key data.

Public Member Functions

- constructor (string pem, __7_ string pass)
  Creates the SSLPrivateKey object from the PEM-encoded text representation of the private key passed.
- constructor (binary der)
  Creates the SSLPrivateKey object from the data argument passed.
- copy ()
  Copying objects of this class is not supported, an exception will be thrown.
- int getBitLength ()
  Returns the bit length of the private key.
- hash getInfo ()
  Returns a hash of all information for the private key.
- string getPEM ()
  Returns a string in PEM format representing the private key.
- string getType ()
  Returns a string giving the algorithm used for the private key.
- int getVersion ()
  Returns a constant value of 1; do not use; only included for backwards-compatibility.

45.51.1 Detailed Description

This class implements a container for private key data.

45.51.2 Member Function Documentation

45.51.2.1 Qore::SSLPrivateKey::constructor (string *pem*, __7_ string *pass*)

Creates the SSLPrivateKey object from the PEM-encoded text representation of the private key passed.

Parameters

pem	The PEM-encoded text representation of the private key
pass	The optional password or passphrase for the private key

Example:

```
my SSLPrivateKey $key($pem);
```

Exceptions

SSLPRIVATEKEY-CONS↩ TRUCTOR-ERROR	error in private key data

Since

0.8.4: the deprecated functionality where the if the string passed was less than 200 bytes long, it was assumed to be a file name has been removed; the string is assumed to be the PEM-encoded private key itself

45.51.2.2 Qore::SSLPrivateKey::constructor (binary *der*)

Creates the SSLPrivateKey object from the data argument passed.

Parameters

der	The DER-encoded binary representation of the private key

Example:

```
my SSLPrivateKey $key($der);
```

Exceptions

SSLPRIVATEKEY-CONS↩ TRUCTOR-ERROR	error in private key data

45.51.2.3 Qore::SSLPrivateKey::copy ( )

Copying objects of this class is not supported, an exception will be thrown.

Exceptions

SSLPRIVATEKEY-COPY-↩ ERROR	SSLPrivateKey objects cannot be copied

45.51.2.4 int Qore::SSLPrivateKey::getBitLength ( )

Returns the bit length of the private key.

Returns

the bit length of the private key

Code Flags:

CONSTANT

Example:

```
my int $len = $key.getBitLength();
```

45.51.2.5 hash Qore::SSLPrivateKey::getInfo ( )

Returns a hash of all information for the private key.

Returns

a hash of all information for the private key with the following keys:

- `"type"`: The type of private key (see SSLPrivateKey::getType())
- `"version"`: The version of the private key (see SSLPrivateKey::getVersion())
- `"bitLength"`: The bit length of the private key (see SSLPrivateKey::getBitLength())

Code Flags:

CONSTANT

Example:

```
my hash $h = $key.getInfo();
```

45.51.2.6 string Qore::SSLPrivateKey::getPEM ( )

Returns a string in PEM format representing the private key.

Returns

a string in PEM format representing the private key

Example:

```
my string $pem = $key.getPEM();
```

Exceptions

SSLPRIVATEKEY-ERROR	could not create PEM string from private key data

45.51.2.7 string Qore::SSLPrivateKey::getType ( )

Returns a string giving the algorithm used for the private key.

Returns

a string giving the algorithm used for the private key (ex: `"RSA"`, `"RSA2"`, `"DSA"`, `"DSA1"`, `"DSA2"`, `"DAS3"`, `"DAS4"`, `"DH"`, `"unknown"`)

Code Flags:

CONSTANT

Example:

```
my string $str = $pkey.getType();
```

45.51.2.8 int Qore::SSLPrivateKey::getVersion ( )

Returns a constant value of 1; do not use; only included for backwards-compatibility.

Code Flags:

CONSTANT

The openssl library never put any usable value into the internal fields that were used to provide this information; newer versions of openssl do not expose this information at all, therefore this method now returns a constant value of 1 for backwards-compatibility.

Returns

a constant value of 1; do not use; only included for backwards-compatibility

45.52 Qore::TermIOS Class Reference

This class allows Qore scripts to get or set terminal settings on UNIX platforms.

Public Member Functions

- constructor ()
  Creates the TermIOS object with random contents.
- copy ()
  Returns a copy of the object.
- int getCC (softint cc)
  Returns the integer value for the given control character from the given control character code.
- int getCFlag ()
  Returns the control mode flag for the object.
- int getIFlag ()
  Returns the input mode flag for the object.
- int getLFlag ()
  Returns the local mode flag for the object.
- int getOFlag ()
  Returns the output mode flag for the object.
- bool isEqual (Termios termios)
  Returns True if the TermIOS object passed as an argument is equal to the current object; False if not.
- nothing setCC (softint offset, softint value)
  Sets the value of the given control character.
- nothing setCFlag (softint flag)

Sets the control mode flag for the object from a mask of Terminal Attribute Control Mode Constants.

- nothing setIFlag (softint flag)

  Sets the input mode flag for the object from a mask of Terminal Attributes Input Mode Constants.

- nothing setLFlag (softint flag)

  Sets the local mode flag for the object from a mask of Terminal Attribute Local Mode Constants.

- nothing setOFlag (softint flag)

  Sets the output mode flag for the object from a mask of Terminal Attributes Output Mode Constants.

Static Public Member Functions

- static hash getWindowSize ()

  Returns a hash giving the current terminal window size in hash keys "`rows`" and "`columns`".

45.52.1 Detailed Description

This class allows Qore scripts to get or set terminal settings on UNIX platforms.

Restrictions:

Qore::PO_NO_TERMINAL_IO

On platforms without TermIOS support (such as Windows), none of the methods in this class are available; to write a portable program, check the Qore::Option::HAVE_TERMIOS constant at runtime before using this functionality.

This class contains the data structure used to read and set terminal attributes on terminal I/O constants (stdin, stdout, stderr).

This class is used with File::getTerminalAttributes(), File::setTerminalAttributes(), and the terminal I/O constants to manipulate terminal attributes.

For example, here is some code to set terminal attributes, read in a character from standard input with a timeout, and reset the terminal attributes:

```
my TermIOS $t();
stdin.getTerminalAttributes($t);
my TermIOS $orig = $t.copy();
on_exit
    stdin.setTerminalAttributes(TCSADRAIN, $orig);

my $lflag = $t.getLFlag();
$lflag &= ~ICANON;
$lflag &= ~ECHO;
$lflag &= ~ISIG;
$t.setLFlag($lflag);
$t.setCC(VMIN, 1);
$t.setCC(VTIME, 0);
stdin.setTerminalAttributes(TCSADRAIN, $t);

stdout.printf("Press any key: ");
while (!stdin.isDataAvailable(20ms)) {
    stdout.printf(".");
    stdout.sync();
    usleep(1ms);
}
my $c = stdin.read(1);
stdout.printf(" GOT ASCII 0x%02x (%d) '%s'\n", ord($c), ord($c), $c);
```

For more information on terminal attributes, see your system's manual pages for "termios".

Note

This class is not available with the PO_NO_TERMINAL_IO parse option.

45.52.2 Member Function Documentation

45.52.2.1 Qore::TermIOS::constructor ( )

Creates the TermIOS object with random contents.

Use File::getTerminalAttributes() with a terminal I/O constant to initialize the object with terminal settings

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my TermIOS $termios();
stdin.getTerminalAttributes($termios);
```

45.52.2.2 Qore::TermIOS::copy ( )

Returns a copy of the object.

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my TermIOS $t2 = $t.copy();
```

45.52.2.3 int Qore::TermIOS::getCC (softint *cc*)

Returns the integer value for the given control character from the given control character code.

Parameters

cc	the control character code for the character to retrieve

Returns

the integer value for the given control character from the given control character code

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
# get minimum characters to return on a read
my int $vmin = $t.getCC(VMIN);
# get character input timer in 0.1 second increments
my int $vtime = $t.getCC(VTIME);
```

Exceptions

TERMIOS-CC-ERROR	the control character is invalid

45.52.2.4 int Qore::TermIOS::getCFlag ( )

Returns the control mode flag for the object.

Returns

the control mode flag for the object

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $flag = $termios.getCFlag();
```

45.52.2.5 int Qore::TermIOS::getIFlag ( )

Returns the input mode flag for the object.

Returns

the input mode flag for the object

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $flag = $termios.getIFlag();
```

45.52.2.6 int Qore::TermIOS::getLFlag ( )

Returns the local mode flag for the object.

Returns

the local mode flag for the object

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $flag = $termios.getLFlag();
```

45.52.2.7 int Qore::TermIOS::getOFlag ( )

Returns the output mode flag for the object.

Returns

the output mode flag for the object

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $flag = $termios.getOFlag();
```

45.52.2.8 static hash Qore::TermIOS::getWindowSize ( ) [static]

Returns a hash giving the current terminal window size in hash keys "rows" and "columns".

Returns

a hash giving the current terminal window size in hash keys "rows" and "columns"

Platform Availability:

Qore::Option::HAVE_TERMIOS

Restrictions:

Qore::PO_NO_TERMINAL_IO

Code Flags:

RET_VALUE_ONLY

Example:

my hash $wh = $termios.getWindowSize();

45.52.2.9 bool Qore::TermIOS::isEqual (Termios *termios*)

Returns True if the TermIOS object passed as an argument is equal to the current object; False if not.

Returns

True if the TermIOS object passed as an argument is equal to the current object; False if not

Platform Availability:

Qore::Option::HAVE_TERMIOS

Code Flags:

CONSTANT

Example:

```
my bool $b = $termios.isEqual($termios2);
```

45.52.2.10 nothing Qore::TermIOS::setCC (softint *offset*, softint *value*)

Sets the value of the given control character.

Parameters

offset	the control character to set
value	the value to set for the control character

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
# set minimum characters to return on a read
$t.setCC(VMIN, 1);

# set character input timer in 0.1 second increments (= no timer)
$t.setCC(VTIME, 0);
```

Exceptions

TERMIOS-CC-ERROR	the control character is invalid

45.52.2.11 nothing Qore::TermIOS::setCFlag (softint *flag*)

Sets the control mode flag for the object from a mask of Terminal Attribute Control Mode Constants.

Parameters

flag	the control mode (see Terminal Attribute Control Mode Constants for masks for this value)

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $cflag = $termios.getCFlag();
$cflag = ($cflag & ~CSIZE) | CS8 | CSTOPB;
$cflag = ($cflag | CLOCAL | CREAD) & ~CRTSCTS;
$termios.setCFlag($cflag);
```

45.52.2.12 nothing Qore::TermIOS::setIFlag (softint *flag*)

Sets the input mode flag for the object from a mask of Terminal Attributes Input Mode Constants.

Parameters

flag	the input mode flag for the object from a mask of Terminal Attributes Input Mode Constants

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $iflag = $termios.getIFlag();
# ring bell on input queue full
$iflag |= IMAXBEL;
$termios.setIFlag($iflag);
```

45.52.2.13 nothing Qore::TermIOS::setLFlag (softint *flag*)

Sets the local mode flag for the object from a mask of Terminal Attribute Local Mode Constants.

Parameters

flag	the local mode (see Terminal Attribute Local Mode Constants for masks for this value)

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $lflag = $termios.getLFlag();
# disable canonical input mode (= turn on "raw" mode)
$lflag &= ~ICANON;
# turn off echo mode
$lflag &= ~ECHO;
# do not check for special input characters (INTR, QUIT, and SUSP)
$lflag &= ~ISIG;
$termios.setLFlag($lflag);
```

45.52.2.14 nothing Qore::TermIOS::setOFlag (softint *flag*)

Sets the output mode flag for the object from a mask of Terminal Attributes Output Mode Constants.

Parameters

flag	the output mode (see Terminal Attributes Output Mode Constants for masks for this value)

Platform Availability:

Qore::Option::HAVE_TERMIOS

Example:

```
my int $oflag = $termios.getOFlag();
# translate linefeeds to crlf
$oflag |= ONLCR;
$termios.setOFlag($oflag);
```

45.53 Qore::Thread::ThreadPool Class Reference

This class defines a thread pool that grows and shrinks dynamically within user-defined limits according to the task load placed on it.

Public Member Functions

- constructor (int max=0, int minidle=0, int maxidle=0, timeout release_ms=5s)

  creates the pool with the given parameters; idle threads are started immediately if necessary
- destructor ()

  destroys the pool; any task threads are detached; to wait for all task threads to complete, call ThreadPool::stopWait() first
- stop ()

  stops the thread pool and returns immediately; after this method has been executed once no more tasks can be submitted to the ThreadPool
- stopWait ()

  stops the thread pool and does not return until all child threads have also been stopped; after this method has been executed once no more tasks can be submitted to the ThreadPool
- submit (code task, __7_ code cancel)

  submit a task to the pool
- string toString ()

  returns a description of the ThreadPool

45.53.1 Detailed Description

This class defines a thread pool that grows and shrinks dynamically within user-defined limits according to the task load placed on it.

The ThreadPool can also pre-allocate idle threads for quickly allocating threads to tasks submitted through ThreadPool::submit() for cases when very low latency is required (for example, for allocating already waiting threads to incoming Socket connections).

A worker thread is started while the ThreadPool is running that waits for tasks in an internal task queue and allocates the tasks to child threads. If an idle thread is available, the task is submitted to that thread immediately, otherwise, if the ThreadPool is not already at maximum capacity (the *max* argument to ThreadPool::constructor()), a new thread is started and the task is allocated to the new thread. Otherwise, the task will block until a thread becomes free, at which time the task is allocated to the newly-freed child thread.

When a child thread has no more tasks to execute, it will either be returned to the pool to wait in an idle state if possible, or it will terminate. Threads are returned to the idle pool if there are fewer than *maxidle* threads in the idle pool already or if there are more tasks in the queue than idle threads.

ThreadPools downscale over time when demand is lower according to the *release_ms* argument to ThreadPool::constructor(); when there more than *minidle* threads in the idle pool, then for each time period defined by *release_ms*, a single thread in the idle pool is released until the number of threads in the idle pool reaches *minidle*.

Therefore the *minidle* argument defines the "ground state" of the ThreadPool (how many idle threads are waiting for tasks), and the *release_ms* argument defines the period in which the ThreadPool returns to its ground state after demand for threads results in a condition where there are temporarily more than *minidle* threads in the idle pool.

If the ThreadPool is stopped when tasks are still in the queue, then any cancellation closure or call reference for the task is executed; see ThreadPool::submit() for more information.

Restrictions:

Qore::PO_NO_THREAD_CLASSES

Example:

```
my ThreadPool $tp(10, 2, 4);
```

Since

Qore 0.8.8

45.53.2 Member Function Documentation

45.53.2.1 Qore::Thread::ThreadPool::constructor (int *max* = 0, int *minidle* = 0, int *maxidle* = 0, timeout *release_ms* = 5s)

creates the pool with the given parameters; idle threads are started immediately if necessary

Example:

```
my ThreadPool $tp(10, 2, 4);
```

When task threads complete, the thread is returned to the pool if there are less than *minidle* threads in the idle list or there are more tasks in the queue than the number of idle threads.

Parameters

max	the maximum number of threads in the pool
minidle	the minimum number of free idle threads to keep ready
maxidle	the maximum number of idle threads to keep ready
release_ms	this value gives the delay in terminating single idle threads when *maxidle* > *minidle* and there are more than *minidle* threads in the idle pool; for example, if *release_ms* = 10s then when there are more than *minidle* threads in the idle pool, every 10 seconds an idle thread is terminated until there are *minidle* threads in the pool. Note that like all Qore functions and methods taking timeout values, a relative date/time value can be used to make the units clear (i.e. 2m = two minutes, etc.)

Exceptions

THREADPOOL-ERROR	minidle > max, maxidle > max or minidle > maxidle, or release_ms < 0

45.53.2.2 Qore::Thread::ThreadPool::destructor ( )

destroys the pool; any task threads are detached; to wait for all task threads to complete, call ThreadPool::stopWait() first

Example:

```
delete $tp;
```

45.53.2.3 Qore::Thread::ThreadPool::stop ( )

stops the thread pool and returns immediately; after this method has been executed once no more tasks can be submitted to the ThreadPool

Example:

```
$tp.stop();
```

This method detaches all child threads immediately, stops the ThreadPool, and returns immediately.

Note

if any task threads are still running; they are detached from the ThreadPool and terminate independently from the ThreadPool

See also

ThreadPool::stopWait()

45.53.2.4 Qore::Thread::ThreadPool::stopWait ( )

stops the thread pool and does not return until all child threads have also been stopped; after this method has been executed once no more tasks can be submitted to the ThreadPool

Example:

```
$tp.stopWait();
```

This method does not return until the ThreadPool has been stopped and all child threads have also been stopped.

See also

ThreadPool::stop()

45.53.2.5 Qore::Thread::ThreadPool::submit (code *task*, *__7_* code *cancel*)

submit a task to the pool

Example:

```
$tp.submit(sub () { call_function($arg); });
```

Parameters

task	the closure or call reference to execute
cancel	an optional closure or call reference to execute if the ThreadPool is stopped before the task can be executed; note that cancellation code is run serially for each task in order of submission in the ThreadPool's worker thread after the ThreadPool has been shut down

45.53.2.6 string Qore::Thread::ThreadPool::toString ( )

returns a description of the ThreadPool

Code Flags:

CONSTANT

Example:

```
my string $desc = $tp.toString();
```

Returns

a description of the ThreadPool

45.54 Qore::TimeZone Class Reference

The TimeZone class provides access to time zone functionality.

Public Member Functions

- int UTCOffset ()

  Returns the number of seconds east of UTC for the zone; negative numbers indicate a zone west of UTC.
- constructor (string region)

  Creates the TimeZone object based on the region name (ex: "America/Chicago")
- constructor (softint seconds_east)

  Creates the TimeZone object based on the number of seconds east of UTC (3600 = UTC +01)
- copy ()

  Creates a copy of the TimeZone object.
- date date (softint secs, softint us=0)

  Returns the equivalent date in the time zone of the current object.
- date date (date d)

  Returns the equivalent date in the time zone of the current object.
- date date (string dtstr)

  Returns the equivalent date in the time zone of the current object.
- date date (string dtstr, string mask)

  Returns a date/time value in the current TimeZone corresponding to parsing a string argument according to a format mask.
- date dateMs (softint ms)

  Returns a date in the object's zone based on an offsets in milliseconds from 1970-01-01Z.
- date dateUs (softint us)

  Returns a date in the object's zone based on an offsets in microseconds from 1970-01-01Z.
- bool hasDST ()

  Returns True if the zone has daylight saving's time rules, False if not.
- string region ()

  Returns the region name as a string; if the current zone is based on a UTC offset, then the UTC offset is returned as a string like "+01:00".

Static Public Member Functions

- static TimeZone get ()

  Returns a TimeZone object for the current time zone.
- static nothing set (TimeZone zone)

  Sets the time zone for the current Program object from a TimeZone object.
- static nothing setRegion (string region)

  Sets the time zone for the current Program object from a time zone region name.
- static nothing setUTCOffset (softint seconds_east)

  Sets the time zone for the current Program object from an integer offset in seconds east of UTC.

45.54.1 Detailed Description

The TimeZone class provides access to time zone functionality.

TimeZone objects based on zoneinfo region files (on UNIX) or registry information (on Windows) can have daylight savings time information; those based on UTC offsets have none.

45.54.2 Member Function Documentation

45.54.2.1 Qore::TimeZone::constructor (string *region*)

Creates the TimeZone object based on the region name (ex: "America/Chicago")

Parameters

region	The region name for the time zone (ex: "America/Chicago"); if the zoneinfo file for the region cannot be found or parsed (on UNIX) or if the registry entry cannot be found (on Windows), then an exception is thrown

Note

On Windows the region names correspond to registry entries under HKEY_LOCAL_MACHINE SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones (ex: "Central Europe Standard Time")

Example:

```
my TimeZone $tz("Europe/Prague");
```

Exceptions

TZINFO-ERROR	Unable to read zoneinfo file; invalid file magic; error parsing zoneinfo file, etc

45.54.2.2 Qore::TimeZone::constructor (softint *seconds_east*)

Creates the TimeZone object based on the number of seconds east of UTC (3600 = UTC +01)

Parameters

seconds_east	The number of seconds east of UTC for the time zone; for zones west of UTC, use negative numbers

Example:

```
my TimeZone $tz(3600);
```

45.54.2.3 Qore::TimeZone::copy ( )

Creates a copy of the TimeZone object.

Example:

```
my TimeZone $newzone = $tz.copy();
```

45.54.2.4 date Qore::TimeZone::date (softint *secs*, softint *us* = 0)

Returns the equivalent date in the time zone of the current object.

Code Flags:

CONSTANT

Parameters

secs	offset are in seconds from 1970-01-01Z
us	offset are in microseconds from 1970-01-01Z

Example:

```
my date $dt = $tz.date($secs);
```

45.54.2.5 date Qore::TimeZone::date (date *d*)

Returns the equivalent date in the time zone of the current object.

Code Flags:

CONSTANT

Parameters

d	A date that will be used to create the date in the time zone of the objects; the same point in time will be returned but in the time zone of the object

Example:

```
my date $dt = $tz.date(2012-01-01T13:56:23+01:00);
```

45.54.2.6 date Qore::TimeZone::date (string *dtstr*)

Returns the equivalent date in the time zone of the current object.

Code Flags:

CONSTANT

Parameters

dtstr	The string to be used to return a date in the object's time zone

Example:

```
my date $dt = $tz.date("2012-01-01T13:56:23");
```

Since

Qore 0.8.4

45.54.2.7 date Qore::TimeZone::date (string *dtstr*, string *mask*)

Returns a date/time value in the current TimeZone corresponding to parsing a string argument according to a format mask.

Code Flags:

RET_VALUE_ONLY

Parameters

dtstr	a string giving a date
mask	the mask for the date value; see Date Mask Format for information on the format of the format mask

Returns

the date/time value in the current TimeZone corresponding to parsing the *dtstr* string argument according to *mask* serving as a format mask

Example:

```
my TimeZone $tz("Europe/London");
printf("%y\n", $tz.date("05/02/2012", "DD/MM/YYYY")); # outputs: 2012-02-05 00:00:00 Sun Z (GMT)
```

Exceptions

DATE-CONVERT-ERROR	invalid mask specification

See also

similar to but more useful than Qore::date(string, string)

Since

Qore 0.8.6

45.54.2.8 date Qore::TimeZone::dateMs (softint *ms*)

Returns a date in the object's zone based on an offsets in milliseconds from `1970-01-01Z`.

Code Flags:

CONSTANT

Parameters

ms	an offsets in milliseconds from `1970-01-01Z`

Returns

a date in the object's zone based on the given offsets in milliseconds from `1970-01-01Z`

Example:

```
my date $dt = $tz.dateMs($offset_ms);
```

45.54.2.9 date Qore::TimeZone::dateUs (softint *us*)

Returns a date in the object's zone based on an offsets in microseconds from `1970-01-01Z`.

Code Flags:

CONSTANT

Parameters

us	an offsets in microseconds from `1970-01-01Z`

Returns

a date in the object's zone based on the given offsets in microseconds from `1970-01-01Z`

Example:

```
my date $dt = $tz.dateUs($offset_uss);
```

45.54.2.10 static TimeZone Qore::TimeZone::get ( ) [static]

Returns a TimeZone object for the current time zone.

Returns

a TimeZone object for the current time zone

Code Flags:

CONSTANT

Example:

```
my TimeZone $tz = TimeZone::get();
```

45.54.2.11 bool Qore::TimeZone::hasDST ( )

Returns True if the zone has daylight saving's time rules, False if not.

TimeZone objects based on zoneinfo region files (on UNIX) or registry information (on Windows) can have (but do not necessarily have) daylight savings time information; those based on UTC offsets have none

Returns

True if the current zone has daylight saving's time rules, False if not

Code Flags:

CONSTANT

Example:

```
my bool $hasdst = $tz.hasDST();
```

45.54.2.12 string Qore::TimeZone::region ( )

Returns the region name as a string; if the current zone is based on a UTC offset, then the UTC offset is returned as a string like `"+01:00"`.

Returns

the region name as a string; if the current zone is based on a UTC offset, then the UTC offset is returned as a string like `"+01:00"`

Code Flags:

CONSTANT

Example:

```
my string $region = $tz.region();
```

45.54.2.13 static nothing Qore::TimeZone::set (TimeZone *zone*) [static]

Sets the time zone for the current Program object from a TimeZone object.

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Parameters

zone	the time zone to set

Example:

```
my TimeZone $tz("Europe/Prague");
TimeZone::set($tz);
```

45.54.2.14 static nothing Qore::TimeZone::setRegion (string *region*) [static]

Sets the time zone for the current Program object from a time zone region name.

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Parameters

region	the region name to set

Note

On Windows the region names correspond to registry entries under `HKEY_LOCAL_MACHINE SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones` (ex: "`Central` Europe Standard Time)

Example:

```
TimeZone::setRegion("Europe/Prague");
```

45.54.2.15 static nothing Qore::TimeZone::setUTCOffset (softint *seconds_east*) [static]

Sets the time zone for the current Program object from an integer offset in seconds east of UTC.

Restrictions:

Qore::PO_NO_LOCALE_CONTROL

Parameters

seconds_east	the number of seconds east of UTC for the new time zone (negative numbers give seconds west of UTC)

Example:

```
TimeZone::setUTCOffset(3600);
```

45.54.2.16 int Qore::TimeZone::UTCOffset ( )

Returns the number of seconds east of UTC for the zone; negative numbers indicate a zone west of UTC.

Returns

the number of seconds east of UTC for the zone; negative numbers indicate a zone west of UTC

Code Flags:

CONSTANT

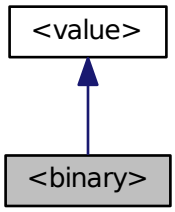
Example:

```
my int $offset = $tz.UTCOffset();
```

45.55 Qore::zzz8binaryzzz9 Class Reference

Methods in this pseudo-class can be executed on binary values.

Inheritance diagram for Qore::zzz8binaryzzz9:



Public Member Functions

- bool empty ()

  Returns True if the binary object is empty (size = 0), False if not.
- int size ()

  Returns the number of bytes in the binary object.
- bool sizep ()

  Returns True since binary objects can return a non-zero size.
- list split (binary sep)

  Returns a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed.
- binary substr (softint start)

  Returns a portion of the binary data starting from an integer offset.
- binary substr (softint start, softint len)

  Returns a portion of the binary data starting from an integer offset.
- string toBase64 (softint maxlinelen=-1)

  Returns the base64-encoded representation of the binary object.
- string toHex ()

  returns a string of hexadecimal digits corresponding to the contents of the binary object
- string toMD5 ()

  Returns the MD5 message digest of the binary data as a hex string.
- string toSHA1 ()

  Returns the SHA1 message digest of the binary data as a hex string.
- string toSHA224 ()

  Returns the SHA-224 message digest (a variant of SHA-2) of the binary data as a hex string.
- string toSHA256 ()

  Returns the SHA-256 message digest (a variant of SHA-2) of the binary data as a hex string.
- string toSHA384 ()

  Returns the SHA-384 message digest (a variant of SHA-2) of the binary data as a hex string.
- string toSHA512 ()

  Returns the SHA-512 message digest (a variant of SHA-2) of the binary data as a hex string.
- string toString (__7_ string encoding)

  Returns a string created from the binary data, taking an optional second argument giving the string encoding; if no second argument is passed then the default character encoding is assumed.
- int typeCode ()

  Returns Qore::NT_BINARY.
- bool val ()

  Returns False if the binary object is empty (size = 0), True if not.

45.55.1 Detailed Description

Methods in this pseudo-class can be executed on binary values.

45.55.2 Member Function Documentation

45.55.2.1 bool Qore::zzz8binaryzzz9::empty ( )

Returns True if the binary object is empty (size = 0), False if not.

The opposite of Qore::zzz8binaryzzz9::val()

Returns

True if the binary object is empty (size = 0), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $bin.empty();
```

45.55.2.2 int Qore::zzz8binaryzzz9::size ( )

Returns the number of bytes in the binary object.

Returns

the number of bytes in the binary object

Code Flags:

CONSTANT

Example:

```
my int $len = $b.size();
```

See also

Qore::zzz8binaryzzz9::sizep()

45.55.2.3 bool Qore::zzz8binaryzzz9::sizep ( )

Returns True since binary objects can return a non-zero size.

Code Flags:

CONSTANT

Returns

True since binary objects can return a non-zero size

See also

Qore::zzz8binaryzzz9::size()

Since

Qore 0.8.9

45.55.2.4 list Qore::zzz8binaryzzz9::split (binary *sep*)

Returns a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed.

Code Flags:

CONSTANT

Parameters

sep	the bytes that separate the fields

Returns

a list of binary objects representing each component of the binary object separated by the bytes identified by the separator argument, with the separator removed

Example:

```
my list $l = $bin.split($sep);
```

Note

equivalent to split(binary, binary)

Since

Qore 0.8.5

45.55.2.5 binary Qore::zzz8binaryzzz9::substr (softint *start*)

Returns a portion of the binary data starting from an integer offset.

Arguments can be negative, giving offsets from the end of the data.

Code Flags:

CONSTANT

Parameters

start	The starting byte for the portion of the data where the first byte is at offset 0; if the offset is negative, it designates the number of bytes from the end of the data

Returns

the portion of the data starting from an integer bte offset; the rest of the data is returned after this offset

Example:

```
# get the last 10 bytes of a binary object
my binary $b1 = $b0.substr(-10);
```

Note

equivalent to Qore::substr(binary, softint)

Since

Qore 0.8.8

45.55.2.6 binary Qore::zzz8binaryzzz9::substr (softint *start*, softint *len*)

Returns a portion of the binary data starting from an integer offset.

Arguments can be negative, giving offsets from the end of the data.

Code Flags:

CONSTANT

Parameters

start	The starting byte for the portion of the data where the first byte is at offset 0; if the offset is negative, it designates the number of bytes from the end of the data
len	The maximum number of characters to copy; if this value is negative, the rest of the string from *start* will be copied to the substring, except without - *len* characters from the end of the string

Returns

the portion of the data starting from an integer bte offset; the rest of the data is returned after this offset

Example:

```
# get the last 5 bytes of the last 10 bytes of a binary object
my binary $b1 = $b0.substr(-10, 5);
```

Note

equivalent to Qore::substr(binary, softint, softint)

Since

Qore 0.8.8

45.55.2.7 string Qore::zzz8binaryzzz9::toBase64 (softint *maxlinelen* = -1)

Returns the base64-encoded representation of the binary object.

Code Flags:

CONSTANT

Example:

```
my string $base64 = $bin.toBase64(64);
```

Implementation based on RFC-1421 and RFC-2045

Parameters

maxlinelen	the maximum length of a line in the resulting output string in bytes; if this value is > 0 then output lines will be separated by CRLF characters

Returns

the base64-encoded string of the data passed

Since

Qore 0.8.8

See also

- Qore::zzz8stringzzz9::toBase64()
- makeBase64String(binary)

45.55.2.8 string Qore::zzz8binaryzzz9::toHex ( )

returns a string of hexadecimal digits corresponding to the contents of the binary object

Code Flags:

CONSTANT

Example:

```
my string $str = $b.toHex();
```

Returns

a string of hexadecimal digits corresponding to the contents of the binary object

Since

Qore 0.8.8

See also

- Qore::zzz8stringzzz9::toHex()
- Qore::makeHexString(binary)

45.55.2.9 string Qore::zzz8binaryzzz9::toMD5 ( )

Returns the `MD5 message digest` of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Returns

a hex string of the digest (ex: "`5d41402abc4b2a76b9719d911017c592`")

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $str = binary("hello").toMD5(); # returns "5d41402abc4b2a76b9719d911017c592"
```

Exceptions

MD5-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

- equivalent to MD5()
- the MD5 algorithm is not collision-resistant; it's recommended to use another hash algorithm (like SHA-256) if cryptographic security is important

See also

- MD5_bin()
- Qore::zzz8stringzzz9::toMD5()

Since

Qore 0.8.5

45.55.2.10 string Qore::zzz8binaryzzz9::toSHA1 ( )

Returns the SHA1 message digest of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Returns

a hex string of the digest (ex: "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d")

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $str = binary("hello").toSHA1(); # returns "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

Exceptions

SHA1-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to SHA1()

See also

- SHA1_bin()
- Qore::zzz8stringzzz9::toSHA1()

Since

Qore 0.8.5

45.55.2.11 string Qore::zzz8binaryzzz9::toSHA224 ( )

Returns the SHA-224 message digest (a variant of SHA-2) of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA224

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193")

Example:

```
my string $str = binary("hello").toSHA224("hello"); # returns "
      ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193"
```

Exceptions

SHA224-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to SHA224()

See also

- SHA224_bin()
- Qore::zzz8stringzzz9::toSHA224()

Since

Qore 0.8.5

45.55.2.12 string Qore::zzz8binaryzzz9::toSHA256 ( )

Returns the SHA-256 message digest (a variant of SHA-2) of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA256

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Example:

```
my string $str = binary("hello").toSHA256(); # returns "
      2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
```

Exceptions

SHA256-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to SHA256()

See also

- SHA256_bin()
- Qore::zzz8stringzzz9::toSHA256()

Since

Qore 0.8.5

45.55.2.13 string Qore::zzz8binaryzzz9::toSHA384 ( )

Returns the SHA-384 message digest (a variant of SHA-2) of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA384

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90

Example:

```
my string $str = binary("hello").toSHA384(); # returns "
      59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90125a3c79f90397bdf5f6a13de828684f"
```

Exceptions

SHA384-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to SHA384()

See also

- SHA384_bin()
- Qore::zzz8stringzzz9::toSHA384()

Since

Qore 0.8.5

45.55.2.14 string Qore::zzz8binaryzzz9::toSHA512 ( )

Returns the SHA-512 message digest (a variant of SHA-2) of the binary data as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA512

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca7

Example:

```
my string $str = binary("hello").toSHA512(); # returns "
      9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3d99ba5c11d7c7acc6e14b8c5da0c4663475c2e5c3adef
```

Exceptions

SHA512-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to SHA512()

See also

- SHA512_bin()
- Qore::zzz8stringzzz9::toSHA512()

Since

Qore 0.8.5

45.55.2.15 string Qore::zzz8binaryzzz9::toString (__7_ string *encoding*)

Returns a string created from the binary data, taking an optional second argument giving the string encoding; if no second argument is passed then the default character encoding is assumed.

Code Flags:

CONSTANT

Parameters

encoding	the character encoding tag for the string return value; if not present, the default character encoding is assumed

Returns

a string created from the binary data passed

Example:

```
my string $str = $b.toString($b, "iso-8859-1");
```

Since

Qore 0.8.8

45.55.2.16 int Qore::zzz8binaryzzz9::typeCode ( )

Returns Qore::NT_BINARY.

Returns

Qore::NT_BINARY

Code Flags:

CONSTANT

Example:

```
switch ($b.typeCode()) {
    case NT_BINARY:
        printf("%y: is a binary\n", $b);
        break;
}
```

45.55.2.17 bool Qore::zzz8binaryzzz9::val ( )

Returns False if the binary object is empty (size = 0), True if not.

The opposite of Qore::zzz8binaryzzz9::empty()

Returns

False if the binary object is empty (size = 0), True if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $bin.val();
```

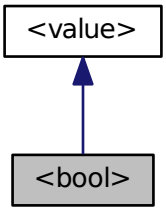
See also

- %perl-bool-eval
- %strict-bool-eval

45.56 Qore::zzz8boolzzz9 Class Reference

Methods in this pseudo-class can be executed on booling-point values.

Inheritance diagram for Qore::zzz8boolzzz9:



Public Member Functions

- bool intp ()

  Returns True because boolean values can be converted to integers (False = 0, True = 1)
- bool strp ()

  Returns True because boolean values can be converted to strings (False = "0", True = "1")
- int typeCode ()

  Returns Qore::NT_BOOLEAN.
- bool val ()

  Returns itself.

45.56.1 Detailed Description

Methods in this pseudo-class can be executed on booling-point values.

45.56.2 Member Function Documentation

45.56.2.1 bool Qore::zzz8boolzzz9::intp ( )

Returns True because boolean values can be converted to integers (False = 0, True = 1)

Returns

True because boolean values can be converted to integers (False = 0, True = 1)

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

45.56.2.2 bool Qore::zzz8boolzzz9::strp ( )

Returns True because boolean values can be converted to strings (False = "0", True = "1")

Returns

True because boolean values can be converted to strings (False = "0", True = "1")

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.56.2.3 int Qore::zzz8boolzzz9::typeCode ( )

Returns Qore::NT_BOOLEAN.

Returns

Qore::NT_BOOLEAN

Code Flags:

CONSTANT

Example:

```
switch ($b.typeCode()) {
    case NT_BOOLEAN:
        printf("%y: is a bool\n", $b);
        break;
}
```

45.56.2.4 bool Qore::zzz8boolzzz9::val ( )

Returns itself.

Returns

itself

Code Flags:

CONSTANT

Example:

```
my bool $bool = $b.val();
```

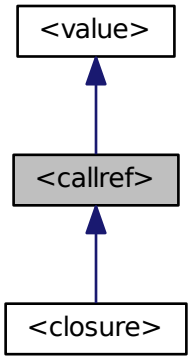
See also

- %perl-bool-eval
- %strict-bool-eval

45.57 Qore::zzz8callrefzzz9 Class Reference

Methods in this pseudo-class can be executed on call references.

Inheritance diagram for Qore::zzz8callrefzzz9:



Public Member Functions

- bool callp ()

  Returns True because this is a callable type.
- any exec (...)

  Evaluates the code with the arguments given and returns the result.
- int typeCode ()

  Returns Qore::NT_CALLREF.
- bool val ()

  Returns True.

45.57.1 Detailed Description

Methods in this pseudo-class can be executed on call references.

45.57.2 Member Function Documentation

45.57.2.1 bool Qore::zzz8callrefzzz9::callp ( )

Returns True because this is a callable type.

Code Flags:

CONSTANT

Example:

```
if ($n.callp())
    printf("the result of calling the value: %y\n", $n());
```

Returns

True because this is a callable type

45.57.2.2 any Qore::zzz8callrefzzz9::exec (...)

Evaluates the code with the arguments given and returns the result.

Parameters

...	the arguments to the code to be called

Returns

the return value of the code called

45.57.2.3 int Qore::zzz8callrefzzz9::typeCode ( )

Returns Qore::NT_CALLREF.

Code Flags:

CONSTANT

Example:

```
switch ($c.typeCode()) {
    case NT_CALLREF:
        printf("%y: is a call reference\n", $c);
        break;
}
```

Returns

Qore::NT_CALLREF

45.57.2.4 bool Qore::zzz8callrefzzz9::val ( )

Returns True.

Code Flags:

CONSTANT

Example:

```
my bool $b = $c.val();
```

Returns

True

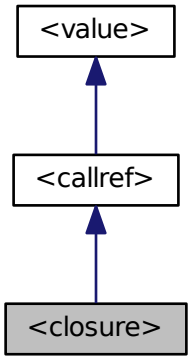
See also

- %perl-bool-eval
- %strict-bool-eval

45.58 Qore::zzz8closurezzz9 Class Reference

Methods in this pseudo-class can be executed on closures.

Inheritance diagram for Qore::zzz8closurezzz9:



Public Member Functions

- int typeCode ()

  Returns Qore::NT_CLOSURE.

45.58.1 Detailed Description

Methods in this pseudo-class can be executed on closures.

45.58.2 Member Function Documentation

45.58.2.1 int Qore::zzz8closurezzz9::typeCode ( )

Returns Qore::NT_CLOSURE.

Returns

Qore::NT_CLOSURE

Code Flags:

CONSTANT

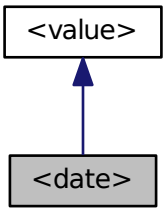
Example:

```
switch ($c.typeCode()) {
    case NT_CLOSURE:
        printf("%y: is a closure\n", $c);
        break;
}
```

45.59 Qore::zzz8datezzz9 Class Reference

Methods in this pseudo-class can be executed on date/time value types.

Inheritance diagram for Qore::zzz8datezzz9:



Public Member Functions

- bool absolute ()

  Returns True if the date is an absolute date/time value.
- __7_ string currentZoneName ()

  Returns the name of the current time zone for the current absolute date/time value (ex: "CEST" for Central European Summer Time for a time during summer time or "CET" for Central European Time for the same time zone while daylight savings time is not active) or a UTC offset (like "+01") or "UTC" or NOTHING for relative date/time values.
- int days ()

  Returns an integer corresponding to the literal day value in the date (does not calculate a duration)
- int durationMicroseconds ()

  Returns an integer value representing the the number of microseconds of time duration in the date value (can be either a relative or absolute date)
- int durationMilliseconds ()

  Returns an integer value representing the the number of milliseconds of time duration in the date value (can be either a relative or absolute date)
- int durationSeconds ()

  Returns an integer value representing the the number of seconds of time duration in the date value (can be either a relative or absolute date)
- string format (string format)

  Returns a formatted string for the date value.
- int getEpochSeconds ()

  Returns the number of seconds since the start of the epoch (1970-01-01Z) for the current date for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values.
- int getEpochSecondsLocalTime ()

  Returns the number of seconds since the start of the epoch (1970-01-01) for the current date in the local time zone for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values.
- int getUtcOffset ()

  Returns the time zone offset for the current time in seconds east of UTC or -1 for relative date/time values.
- int hours ()

  Returns an integer corresponding to the literal hour value in the date (does not calculate a duration)
- hash info ()

  Returns a hash of broken-down date/time information for the date (can be either a relative or absolute date)

- bool intp ()

  Returns True because date values can be converted to integers.
- bool isDst ()

  Returns True if the current date/time value is currently in daylight savings time.
- int microseconds ()

  Returns an integer corresponding to the literal microsecond value in the date (does not calculate a duration)
- date midnight ()

  Returns midnight on the given date (strips the time component on the new value)
- int milliseconds ()

  Returns an integer corresponding to the literal millisecond value in the date (does not calculate a duration)
- int minutes ()

  Returns an integer corresponding to the literal minute value in the date (does not calculate a duration)
- int months ()

  Returns an integer corresponding to the literal month value in the date (does not calculate a duration)
- bool relative ()

  Returns True if the date is a relative date/time value.
- int seconds ()

  Returns an integer corresponding to the literal second value in the date (does not calculate a duration)
- bool strp ()

  Returns True because boolean values can be converted to strings.
- int typeCode ()

  Returns Qore::NT_DATE.
- bool val ()

  Returns False if the date value is all zeros, True if not.
- int years ()

  Returns an integer corresponding to the literal year value in the date (does not calculate a duration)
- __7_ TimeZone zone ()

  Returns a Qore::TimeZone object for the time zone of the date/time value; returns NOTHING for relative date/time values.

45.59.1 Detailed Description

Methods in this pseudo-class can be executed on date/time value types.

45.59.2 Member Function Documentation

45.59.2.1 bool Qore::zzz8datezzz9::absolute ( )

Returns True if the date is an absolute date/time value.

Returns

True if the date is an absolute date/time value

Example:

```
my bool $b = $d.absolute();
```

45.59.2.2 __7_ string Qore::zzz8datezzz9::currentZoneName ( )

Returns the name of the current time zone for the current absolute date/time value (ex: "CEST" for Central European Summer Time for a time during summer time or "CET" for Central European Time for the same time zone while daylight savings time is not active) or a UTC offset (like "+01") or "UTC" or NOTHING for relative date/time values.

Returns

the name of the current time zone for the current absolute date/time value (ex: "CEST" for Central European Summer Time for a time during summer time or "CET" for Central European Time for the same time zone while daylight savings time is not active) or a UTC offset (like "+01") or "UTC" or NOTHING for relative date/time values

Code Flags:

CONSTANT

Example:

```
my *string $zn = $d.currentZoneName();
```

45.59.2.3 int Qore::zzz8datezzz9::days ( )

Returns an integer corresponding to the literal day value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal day value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.days();
```

Note

equivalent to get_days(date)

45.59.2.4 int Qore::zzz8datezzz9::durationMicroseconds ( )

Returns an integer value representing the the number of microseconds of time duration in the date value (can be either a relative or absolute date)

Returns

an integer value representing the the number of microseconds in the date value; if the value is a relative date, the value is normalized to microseconds and the integer microseconds value is returned, if the value is an absolute date, the duration in microseconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of microseconds from the present time to the future date is returned, if an absolute date in the past is used, also 0 is returned (the pseudo-method does not calculate microsecond differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Code Flags:

CONSTANT

Example:

```
my int $us = PT2M15S3u.durationMicroseconds(); # returns 135000003
```

Note

- equivalent to get_duration_microseconds()
- to get the literal microseconds integer value from a date/time value without calculating a duration, use Qore::zzz8datezzz9::microseconds()

See also

- Qore::zzz8datezzz9::durationSeconds()
- Qore::zzz8datezzz9::durationMilliseconds()

Since

Qore 0.8.7

45.59.2.5 int Qore::zzz8datezzz9::durationMilliseconds ( )

Returns an integer value representing the the number of milliseconds of time duration in the date value (can be either a relative or absolute date)

The duration in milliseconds is calculated and any fractional milliseconds are truncated (no rounding is performed)

Returns

an integer value representing the the number of milliseconds in the date value; if the value is a relative date, the value is normalized to milliseconds and the integer milliseconds value is returned, if the value is an absolute date, the duration in milliseconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of milliseconds from the present time to the future date is returned, if an absolute date in the past is used, also 0 is returned (the pseudo-method does not calculate millisecond differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Code Flags:

CONSTANT

Example:

```
my int $us = PT2M15S3u.durationMilliseconds(); # returns 135000
```

Note

- equivalent to get_duration_milliseconds()
- to get the literal milliseconds integer value from a date/time value without calculating a duration, use Qore::zzz8datezzz9::milliseconds()

See also

- Qore::zzz8datezzz9::durationSeconds()
- Qore::zzz8datezzz9::durationMicroseconds()

Since

Qore 0.8.7

45.59.2.6 int Qore::zzz8datezzz9::durationSeconds ( )

Returns an integer value representing the the number of seconds of time duration in the date value (can be either a relative or absolute date)

The duration in seconds is calculated and any fractional seconds are truncated (no rounding is performed)

Returns

an integer value representing the the number of seconds in the date value; if the value is a relative date, the value is normalized to seconds and the integer seconds value is returned, if the value is an absolute date, the duration in seconds is calculated from the present time; so if the present time is sent as an argument, 0 is returned, if a future date is used, the number of seconds from the present time to the future date is returned, if an absolute date in the past is used, also 0 is returned (the pseudo-method does not calculate second differences for absolute dates in the past (for this use Date/Time Arithmetic instead); this function can only return a negative value if passed a relative date/time value

Code Flags:

CONSTANT

Example:

```
my int $us = PT2M15S3u.durationSeconds(); # returns 135
```

Note

- equivalent to get_duration_seconds()
- to get the literal seconds integer value from a date/time value without calculating a duration, use Qore::zzz8datezzz9::seconds()

See also

- Qore::zzz8datezzz9::durationMilliseconds()
- Qore::zzz8datezzz9::durationMicroseconds()

Since

Qore 0.8.7

45.59.2.7 string Qore::zzz8datezzz9::format (string *format*)

Returns a formatted string for the date value.

Code Flags:

CONSTANT

Parameters

format	a string giving the format for the date; see Date Formatting Codes for more information about this string

Returns

a formatted string for a date argument passed

Example:

```
my string $str = $d.format("Day, Mon D, YYYY-MM-DD HH:mm:SS");
```

Bug there is no locale support; day and month names and abbreviations are only returned in English

Note

equivalent to format_date(string, date)

45.59.2.8 int Qore::zzz8datezzz9::getEpochSeconds ( )

Returns the number of seconds since the start of the epoch (1970-01-01Z) for the current date for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values.

Code Flags:

CONSTANT

Example:

```
my int $secs = $d.getEpochSeconds();
```

Returns

the number of seconds since the start of the epoch (1970-01-01Z) for the current date for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values

Since

Qore 0.8.8

45.59.2.9 int Qore::zzz8datezzz9::getEpochSecondsLocalTime ( )

Returns the number of seconds since the start of the epoch (1970-01-01) for the current date in the local time zone for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values.

Code Flags:

CONSTANT

Example:

```
my int $secs = $d.getEpochSecondsLocalTime();
```

Returns

the number of seconds since the start of the epoch (1970-01-01) for the current date in the local time zone for ref absolute_dates "absolute date/time values"; returns 0 for relative date/time values

Since

Qore 0.8.8

45.59.2.10 int Qore::zzz8datezzz9::getUtcOffset ( )

Returns the time zone offset for the current time in seconds east of UTC or -1 for relative date/time values.

Returns

the time zone offset for the current time in seconds east of UTC or -1 for relative date/time values

Code Flags:

CONSTANT

Example:

```
my int $utcoffset = $d.getUtcOffset();
```

45.59.2.11 int Qore::zzz8datezzz9::hours ( )

Returns an integer corresponding to the literal hour value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal hour value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.hours();
```

Note

equivalent to get_hours(date)

45.59.2.12 hash Qore::zzz8datezzz9::info ( )

Returns a hash of broken-down date/time information for the date (can be either a relative or absolute date)

Code Flags:

CONSTANT

Example:

```
my hash $h = date_info($date);
```

Returns

a hash of broken-down date/time information for the given date argument

Note

equivalent to date_info(date)

Since

Qore 0.8.8

45.59.2.13 bool Qore::zzz8datezzz9::intp ( )

Returns True because date values can be converted to integers.

Returns

True because date values can be converted to integers

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

45.59.2.14 bool Qore::zzz8datezzz9::isDst ( )

Returns True if the current date/time value is currently in daylight savings time.

Returns

True if the current date/time value is currently in daylight savings time; always returns False for relative date/time values

Code Flags:

CONSTANT

Example:

```
my bool $b = $d.isDst();
```

45.59.2.15 int Qore::zzz8datezzz9::microseconds ( )

Returns an integer corresponding to the literal microsecond value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal microsecond value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.microseconds();
```

Note

- equivalent to get_microseconds(date)
- to get the number of microseconds of duration in a date/time value, use Qore::zzz8datezzz9::durationMicroseconds() instead

45.59.2.16 date Qore::zzz8datezzz9::midnight ( )

Returns midnight on the given date (strips the time component on the new value)

Returns

midnight on the given date (strips the time component on the new value)

Code Flags:

CONSTANT

Example:

```
my date $midnight = $dt.midnight();
```

Note

equivalent to get_midnight()

Since

Qore 0.8.5

45.59.2.17 int Qore::zzz8datezzz9::milliseconds ( )

Returns an integer corresponding to the literal millisecond value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal millisecond value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.milliseconds();
```

Note

- equivalent to get_milliseconds(date)
- to get the number of milliseconds of duration in a date/time value, use Qore::zzz8datezzz9::durationMilliseconds() instead

45.59.2.18 int Qore::zzz8datezzz9::minutes ( )

Returns an integer corresponding to the literal minute value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal minute value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.minutes();
```

Note

equivalent to get_minutes(date)

45.59.2.19 int Qore::zzz8datezzz9::months ( )

Returns an integer corresponding to the literal month value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal month value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.months();
```

Note

equivalent to get_months(date)

45.59.2.20 bool Qore::zzz8datezzz9::relative ( )

Returns True if the date is a relative date/time value.

Returns

True if the date is a relative date/time value

Example:

```
my bool $b = $d.relative();
```

45.59.2.21 int Qore::zzz8datezzz9::seconds ( )

Returns an integer corresponding to the literal second value in the date (does not calculate a duration)

The date value can be either a relative or absolute date.

Returns

an integer corresponding to the literal second value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.seconds();
```

Note

- equivalent to get_seconds(date)
- to get the number of seconds of duration in a date/time value, use Qore::zzz8datezzz9::durationSeconds() instead

45.59.2.22 bool Qore::zzz8datezzz9::strp ( )

Returns True because boolean values can be converted to strings.

Returns

True because boolean values can be converted to strings

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.59.2.23 int Qore::zzz8datezzz9::typeCode ( )

Returns Qore::NT_DATE.

Returns

Qore::NT_DATE

Example:

```
switch ($d.typeCode()) {
    case NT_DATE:
        printf("%y: is a date\n", $d);
        break;
}
```

45.59.2.24 bool Qore::zzz8datezzz9::val ( )

Returns False if the date value is all zeros, True if not.

Returns

False if the date value is all zeros, True if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $d.val();
```

See also

- %perl-bool-eval
- %strict-bool-eval

45.59.2.25 int Qore::zzz8datezzz9::years ( )

Returns an integer corresponding to the literal year value in the date (does not calculate a duration)

Returns

an integer corresponding to the literal year value in the date (does not calculate a duration)

Code Flags:

CONSTANT

Example:

```
my int $n = $d.years();
```

Note

equivalent to get_years(date)

45.59.2.26 __7_ TimeZone Qore::zzz8datezzz9::zone ( )

Returns a Qore::TimeZone object for the time zone of the date/time value; returns NOTHING for relative date/time values.

Returns

a Qore::TimeZone object for the time zone of the date/time value; returns NOTHING for relative date/time values

Code Flags:

CONSTANT

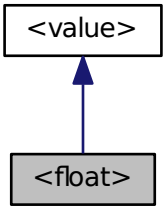
Example:

```
my *TimeZone $zone = $d.zone();
```

45.60 Qore::zzz8floatzzz9 Class Reference

Methods in this pseudo-class can be executed on floating-point values.

Inheritance diagram for Qore::zzz8floatzzz9:



Public Member Functions

- float abs ()

  Returns the absolute value of the number.
- string format (string fmt)

  Returns a string of a formatted number according to a format string.
- bool intp ()

  Returns True because float values can be converted to integers.
- int sign ()

  Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.
- bool strp ()

  Returns True because float values can be converted to strings.
- int typeCode ()

  Returns Qore::NT_FLOAT.
- bool val ()

  Returns True if the float is non-zero, False if zero.

45.60.1 Detailed Description

Methods in this pseudo-class can be executed on floating-point values.

45.60.2 Member Function Documentation

45.60.2.1 float Qore::zzz8floatzzz9::abs ( )

Returns the absolute value of the number.

Code Flags:

CONSTANT

Example:

```
$f = $f.abs();
```

Returns

the absolute value of the number

Note

equivalent to abs(float)

Since

Qore 0.8.8

45.60.2.2 string Qore::zzz8floatzzz9::format (string *fmt*)

Returns a string of a formatted number according to a format string.

Code Flags:

CONSTANT

Parameters

fmt	the format string has the following format: <*thousands_separator*> [<*decimal_separator*><*decimals*>] where: • *thousands_separator* and *decimal_separator* are single ASCII characters defining the thousands and decimal separator characters respectively, and • *decimals* is a single digit defining how may decimals should appear after the decimal point

Returns

a string of a formatted number according to a format string; if the format string does not follow the given format, then an empty string is returned

Example:

```
my float $f = -48392093894.2349;
my string $nstr = $f.format(".,3"); # returns "-48.392.093.894,235"
```

Note

equivalent to format_number(string, softfloat)

Since

Qore 0.8.6

45.60.2.3 bool Qore::zzz8floatzzz9::intp ( )

Returns True because float values can be converted to integers.

Returns

True because float values can be converted to integers

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

45.60.2.4 int Qore::zzz8floatzzz9::sign ( )

Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.

Returns

-1 if the number is negative, 0 if it is zero, or 1 if it is positive

Code Flags:

CONSTANT

Example:

```
printf("sign: %d\n", $f.sign());
```

Since

Qore 0.8.6

45.60.2.5 bool Qore::zzz8floatzzz9::strp ( )

Returns True because float values can be converted to strings.

Returns

True because float values can be converted to strings

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.60.2.6 int Qore::zzz8floatzzz9::typeCode ( )

Returns Qore::NT_FLOAT.

Returns

Qore::NT_FLOAT

Code Flags:

CONSTANT

Example:

```
switch ($f.typeCode()) {
    case NT_FLOAT:
        printf("%y: is a float\n", $f);
        break;
}
```

45.60.2.7 bool Qore::zzz8floatzzz9::val ( )

Returns True if the float is non-zero, False if zero.

Returns

True if the float is non-zero, False if zero

Code Flags:

CONSTANT

Example:

```
my bool $b = $f.val();
```

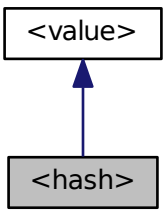
See also

- %perl-bool-eval
- %strict-bool-eval

45.61 Qore::zzz8hashzzz9 Class Reference

Methods in this pseudo-class can be executed on hash values.

Inheritance diagram for Qore::zzz8hashzzz9:



Public Member Functions

- bool compareKeys (hash oh)

  Returns True if the hash argument passed has the same keys in the same order as the current hash, False if not.
- HashListIterator contextIterator ()

  Returns a HashListIterator object for the hash.
- bool empty ()

  Returns True if the hash has no keys, False if it does.
- __7_ string firstKey ()

  Returns the first key name in the hash or NOTHING if the hash has no keys.
- any firstValue ()

  Returns the value assigned to the first key in the hash if any or NOTHING if the hash has no keys.
- bool hasKey (softstring key)

  Returns True if the key exists in the hash (may or may not be assigned a value), False if not.
- bool hasKeyValue (softstring key)

  Returns True if the key exists and is assigned to a value, False if not.
- AbstractIterator iterator ()

  Returns a HashIterator object for the hash.
- HashKeyIterator keyIterator ()

  Returns a HashKeyIterator object for the hash.
- list keys ()

  Returns a list of key names of the hash.
- __7_ string lastKey ()

  Returns the last key name in the hash or NOTHING if the hash has no keys.
- any lastValue ()

  Returns the value assigned to the last key in the hash if any or NOTHING if the hash has no keys.
- HashPairIterator pairIterator ()

  Returns a HashPairIterator object for the hash.
- int size ()

  Returns the number of keys in the hash.
- bool sizep ()

  Returns True since hashes can return a non-zero size.
- int typeCode ()

  Returns Qore::NT_HASH.
- bool val ()

  Returns False if the hash has no keys, True if it does.
- list values ()

  Returns a list of values of the hash.

45.61.1 Detailed Description

Methods in this pseudo-class can be executed on hash values.

45.61.2 Member Function Documentation

45.61.2.1 bool Qore::zzz8hashzzz9::compareKeys (hash *oh*)

Returns True if the hash argument passed has the same keys in the same order as the current hash, False if not.

Code Flags:

CONSTANT

Example:

```
my bool $b = $h.compareKeys($oh);
```

Values are not compraed, only the key names and order

Parameters

oh	a hash to compare the keys against

Returns

True if the hash argument passed has the same keys in the same order as the current hash, False if not

Since

Qore 0.8.8

45.61.2.2 HashListIterator Qore::zzz8hashzzz9::contextIterator ( )

Returns a HashListIterator object for the hash.

Returns

a HashListIterator object for the hash

Code Flags:

CONSTANT

Example:

```
my *hash $q = $db.select("select * from table_name");
map printf("+ %s\n", $1), $h.contextIterator();
```

Note

- this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)
- this pseudo-method is very useful when iterating query results from Datasource::select() or DatasourcePool::select() with the map operator, for example

Since

Qore 0.8.6.2

45.61.2.3 bool Qore::zzz8hashzzz9::empty ( )

Returns True if the hash has no keys, False if it does.

Returns

True if the hash has no keys, False if it does

Code Flags:

CONSTANT

Example:

```
my bool $b = $h.empty();
```

Note

this pseudo-method is also implemented in Qore::zzz8valuezzz9, so it's safe to use on all data types

45.61.2.4 __7_ string Qore::zzz8hashzzz9::firstKey ( )

Returns the first key name in the hash or NOTHING if the hash has no keys.

Returns

the first key name in the hash or NOTHING if the hash has no keys

Code Flags:

CONSTANT

Example:

```
my *string $n = $h.firstKey();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::lastKey()

45.61.2.5 any Qore::zzz8hashzzz9::firstValue ( )

Returns the value assigned to the first key in the hash if any or NOTHING if the hash has no keys.

Returns

the value assigned to the first key in the hash if any or NOTHING if the hash has no keys

Code Flags:

CONSTANT

Example:

```
my any $v = $h.firstValue();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::lastValue()

45.61.2.6 bool Qore::zzz8hashzzz9::hasKey (softstring *key*)

Returns True if the key exists in the hash (may or may not be assigned a value), False if not.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the key name to check

Returns

True if the key exists in the hash (may or may not be assigned a value), False if not

Example:

```
my bool $b = $h.hasKey($key);
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::hasKeyValue(softstring)

45.61.2.7 bool Qore::zzz8hashzzz9::hasKeyValue (softstring *key*)

Returns True if the key exists and is assigned to a value, False if not.

Code Flags:

RET_VALUE_ONLY

Parameters

key	the key name to check

Returns

True if the key exists and is assigned to a value, False if not

Example:

```
my bool $b = $h.hasKeyValue($key);
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::hasKey(softstring)

45.61.2.8 AbstractIterator Qore::zzz8hashzzz9::iterator ( )

Returns a HashIterator object for the hash.

Returns

a HashIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %s: %y\n", $1.key, $1.value), $h.iterator();
```

Note

this pseudo-method is also implemented in Qore::zzz8valuezzz9, so it's safe to use on all data types

Since

Qore 0.8.6

45.61.2.9 HashKeyIterator Qore::zzz8hashzzz9::keyIterator ( )

Returns a HashKeyIterator object for the hash.

Returns

a HashKeyIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %s\n", $1), $h.keyIterator();
```

Note

- this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)
- this pseudo-method is very useful when using a hash as a simulation for a set of strings and quickly iterating the hash with the map operator, for example

Since

Qore 0.8.6.2

45.61.2.10 list Qore::zzz8hashzzz9::keys ( )

Returns a list of key names of the hash.

Returns

a list of key names of the hash

Code Flags:

CONSTANT

Example:

```
my list $l = $h.keys();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::values()

45.61.2.11 __7_ string Qore::zzz8hashzzz9::lastKey ( )

Returns the last key name in the hash or NOTHING if the hash has no keys.

Returns

the last key name in the hash or NOTHING if the hash has no keys

Code Flags:

CONSTANT

Example:

```
my *string $n = $h.lastKey();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::firstKey()

45.61.2.12 any Qore::zzz8hashzzz9::lastValue ( )

Returns the value assigned to the last key in the hash if any or NOTHING if the hash has no keys.

Returns

the value assigned to the last key in the hash if any or NOTHING if the hash has no keys

Code Flags:

CONSTANT

Example:

```
my any $v = $h.lastValue();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

See also

Qore::zzz8hashzzz9::firstValue()

45.61.2.13 HashPairIterator Qore::zzz8hashzzz9::pairIterator ( )

Returns a HashPairIterator object for the hash.

Returns

a HashPairIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %s\n", $1), $h.pairIterator();
```

Note

this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

Since

Qore 0.8.6.2

45.61.2.14 int Qore::zzz8hashzzz9::size ( )

Returns the number of keys in the hash.

The opposite of Qore::zzz8hashzzz9::val()

Returns

the number of keys in the hash

Code Flags:

CONSTANT

Example:

```
my int $num = $h.size();
```

Note

this pseudo-method is also implemented in Qore::zzz8valuezzz9, so it's safe to use on all data types

See also

Qore::zzz8hashzzz9::sizep()

45.61.2.15 bool Qore::zzz8hashzzz9::sizep ( )

Returns True since hashes can return a non-zero size.

Code Flags:

CONSTANT

Returns

True since hashes can return a non-zero size

See also

Qore::zzz8hashzzz9::size()

Since

Qore 0.8.9

45.61.2.16 int Qore::zzz8hashzzz9::typeCode ( )

Returns Qore::NT_HASH.

Returns

Qore::NT_HASH

Code Flags:

CONSTANT

Example:

```
switch ($h.typeCode()) {
    case NT_HASH:
        printf("%y: is a hash\n", $h);
        break;
}
```

Note

this pseudo-method is also implemented in Qore::zzz8valuezzz9, so it's safe to use on all data types

45.61.2.17 bool Qore::zzz8hashzzz9::val ( )

Returns False if the hash has no keys, True if it does.

The opposite of Qore::zzz8hashzzz9::empty()

Returns

False if the hash has no keys, True if it does

Code Flags:

CONSTANT

Example:

```
my bool $b = $h.val();
```

Note

this pseudo-method is also implemented in Qore::zzz8valuezzz9, so it's safe to use on all data types

See also

- %perl-bool-eval
- %strict-bool-eval

45.61.2.18 list Qore::zzz8hashzzz9::values ( )

Returns a list of values of the hash.

Returns

a list of values of the hash

Code Flags:

CONSTANT

Example:

```
my list $l = $h.values();
```

Note

- equivalent to hash_values()
- this pseudo-method is also implemented in Qore::zzz8nothingzzz9, so it's safe to use where the value could be a hash or NOTHING (ex: *hash)

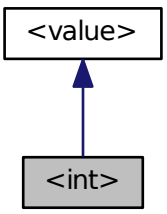
See also

Qore::zzz8hashzzz9::keys()

45.62 Qore::zzz8intzzz9 Class Reference

Methods in this pseudo-class can be executed on integer values.

Inheritance diagram for Qore::zzz8intzzz9:



Public Member Functions

- int abs ()

  Returns the absolute value of the number.
- binary encodeLsb (int size=4)

  returns a binary object with the integer encoded in the given number of bytes in least significant byte order
- binary encodeMsb (int size=4)

  returns a binary object with the integer encoded in the given number of bytes in most significant byte order
- string format (string fmt)

  Returns a string of a formatted number according to a format string.
- bool intp ()

  Returns True by default.
- int sign ()

  Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.
- bool strp ()

  Returns True because integer values can be converted to strings.
- string toUnicode ()

  Returns a single character string in UTF-8 encoding for the integer value treated as a unicode value.
- int typeCode ()

  Returns Qore::NT_INT.
- bool val ()

  Returns True if the int is non-zero, False if zero.

45.62.1 Detailed Description

Methods in this pseudo-class can be executed on integer values.

45.62.2 Member Function Documentation

45.62.2.1 int Qore::zzz8intzzz9::abs ( )

Returns the absolute value of the number.

Code Flags:

CONSTANT

Example:

```
$i = $i.abs();
```

Returns

the absolute value of the number

Note

equivalent to abs(int)

Since

Qore 0.8.8

45.62.2.2 binary Qore::zzz8intzzz9::encodeLsb (int *size* = 4)

returns a binary object with the integer encoded in the given number of bytes in least significant byte order

Code Flags:

RET_VALUE_ONLY

Example:

```
$b += $i.encodeLsb(8);
```

Parameters

size	the size of the encoded binary object; must be 1, 2, 4, or 8 or a SIZE-ERROR exception is raised

Returns

a binary object with the integer encoded in the given number of bytes in least significant byte order

Exceptions

SIZE-ERROR	the argument is not 1, 2, 4, or 8 or the argument cannot fit in the number of bytes requested

Since

Qore 0.8.8

45.62.2.3 binary Qore::zzz8intzzz9::encodeMsb (int *size* = 4)

returns a binary object with the integer encoded in the given number of bytes in most significant byte order

Code Flags:

RET_VALUE_ONLY

Example:

```
$b += $i.encodeMsb(8);
```

Parameters

size	the size of the encoded binary object; must be 1, 2, 4, or 8 or a SIZE-ERROR exception is raised

Returns

a binary object with the integer encoded in the given number of bytes in most significant byte order

Exceptions

SIZE-ERROR	the argument is not 1, 2, 4, or 8 or the argument cannot fit in the number of bytes requested

Since

Qore 0.8.8

45.62.2.4 string Qore::zzz8intzzz9::format (string *fmt*)

Returns a string of a formatted number according to a format string.

Code Flags:

CONSTANT

Parameters

fmt	the format string has the following format: <*thousands_separator*>[<*decimal_separator*><*decimals*>] where: • *thousands_separator* and *decimal_separator* are single ASCII characters defining the thousands and decimal separator characters respectively, and • *decimals* is a single digit defining how may decimals should appear after the decimal point - if this is non-zero then the decimals will all be "0"

Returns

a string of a formatted number according to a format string; if the format string does not follow the given format, then an empty string is returned

Example:

```
my int $i = -48392093894;
my string $nstr = $i.format(".,3"); # returns "-48.392.093.894,000"
```

See also

- Qore::zzz8floatzzz9::format(string)
- Qore::zzz8numberzzz9::format(string)

Since

Qore 0.8.6

45.62.2.5 bool Qore::zzz8intzzz9::intp ( )

Returns True by default.

Returns

True by default

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

45.62.2.6 int Qore::zzz8intzzz9::sign ( )

Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.

Returns

-1 if the number is negative, 0 if it is zero, or 1 if it is positive

Code Flags:

CONSTANT

Example:

```
printf("sign: %d\n", $i.sign());
```

Since

Qore 0.8.6

45.62.2.7 bool Qore::zzz8intzzz9::strp ( )

Returns True because integer values can be converted to strings.

Returns

True because integer values can be converted to strings

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.62.2.8 string Qore::zzz8intzzz9::toUnicode ( )

Returns a single character string in UTF-8 encoding for the integer value treated as a unicode value.

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $str = 0x00f8.toUnicode()
```

Returns

a single character string in UTF-8 encoding for the integer value treated as a unicode value

Note

only the least-significant 4 bytes of the integer are used

Since

Qore 0.8.8

45.62.2.9 int Qore::zzz8intzzz9::typeCode ( )

Returns Qore::NT_INT.

Returns

Qore::NT_INT

Code Flags:

CONSTANT

Example:

```
switch ($i.typeCode()) {
    case NT_INT:
        printf("%y: is a int\n", $i);
        break;
}
```

45.62.2.10 bool Qore::zzz8intzzz9::val ( )

Returns True if the int is non-zero, False if zero.

Returns

True if the int is non-zero, False if zero

Code Flags:

CONSTANT

Example:

```
my bool $b = $i.val();
```

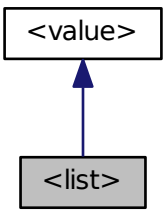
See also

- %perl-bool-eval
- %strict-bool-eval

45.63 Qore::zzz8listzzz9 Class Reference

Methods in this pseudo-class can be executed on lists.

Inheritance diagram for Qore::zzz8listzzz9:



Public Member Functions

- bool contains (any arg)

  Returns True if the list contains arg, False if it does not.
- bool empty ()

  Returns True if the list is empty (size = 0), False if not.
- any first ()

  Returns the first entry in the list.
- AbstractIterator iterator ()

  Returns a ListIterator object for the list.
- string join (string str)

  Creates a string from the list and a separator string given as an argument.
- any last ()

  Returns the last entry in the list.
- int lsize ()

  Returns the number of elements in the list.
- AbstractIterator rangeIterator ()

  Returns a RangeIterator object for the list elements.
- int size ()

  Returns the number of elements in the list.
- bool sizep ()

  Returns True since lists can return a non-zero size.
- int typeCode ()

  Returns Qore::NT_LIST.
- bool val ()

  Returns False if the list is empty (size = 0), True if not.

45.63.1 Detailed Description

Methods in this pseudo-class can be executed on lists.

45.63.2 Member Function Documentation

45.63.2.1 bool Qore::zzz8listzzz9::contains (any *arg*)

Returns True if the list contains arg, False if it does not.

This call uses "soft" comparisons (where types may be converted). The computational complexity is `O(n)` (n = the length of the list).

Code Flags:

RET_VALUE_ONLY

Parameters

arg	any value to check its presence in the list

Returns

True if the list contains arg, False if it does not

Example:

```
my list $l = (1, 2, 3);
my bool $b = $l.contains(5); # returns False
```

45.63.2.2 bool Qore::zzz8listzzz9::empty ( )

Returns True if the list is empty (size = 0), False if not.

The opposite of Qore::zzz8listzzz9::val()

Returns

True if the list is empty (size = 0), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $l.empty();
```

45.63.2.3 any Qore::zzz8listzzz9::first ( )

Returns the first entry in the list.

Returns

the first entry in the list

Code Flags:

CONSTANT

Example:

```
my any $e = $l.first();
```

See also

Qore::zzz8listzzz9::last()

Since

Qore 0.8.5

45.63.2.4 AbstractIterator Qore::zzz8listzzz9::iterator ( )

Returns a ListIterator object for the list.

Returns

a ListIterator object for the list

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), $l.iterator();
```

Since

Qore 0.8.6

45.63.2.5 string Qore::zzz8listzzz9::join (string *str*)

Creates a string from the list and a separator string given as an argument.

Each element in the list will be converted to a string if necessary to be concatenated to the return value string; additionally if any string argument has a different character encoding than *str*, then it will be converted to *str*'s character encoding before concatentation to the return value string

Code Flags:

RET_VALUE_ONLY

Parameters

str	the separator string

Returns

a string created from a list and a separator string, each element in the list will be present in the return value separated by the separator string; the string returned with have the same character encoding as *str*

Example:

```
my string $str = ("a", "b", "c").join(":"); # returns "a:b:c"
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

Note

equivalent to join(string, list)

Since

Qore 0.8.5

45.63.2.6 any Qore::zzz8listzzz9::last ( )

Returns the last entry in the list.

Returns

the last entry in the list

Code Flags:

CONSTANT

Example:

```
my any $e = $l.last();
```

See also

Qore::zzz8listzzz9::first()

Since

Qore 0.8.5

45.63.2.7 int Qore::zzz8listzzz9::lsize ( )

Returns the number of elements in the list.

For this type, this method is equivalent to size()

Returns

the number of elements in the list

Code Flags:

CONSTANT

Example:

```
printf("iterating %d element%s\n", $val.lsize(), $val.lsize() == 1 ? "" : "s");
foreach my any $element in ($val) {
    do_something($element);
}
```

45.63.2.8 AbstractIterator Qore::zzz8listzzz9::rangeIterator ( )

Returns a RangeIterator object for the list elements.

Returns

a RangeIterator object for the list elements

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), $l.ranegIterator();
```

Since

Qore 0.8.8

45.63.2.9 int Qore::zzz8listzzz9::size ( )

Returns the number of elements in the list.

Returns

the number of elements in the list

Code Flags:

CONSTANT

Example:

```
my int $len = $l.size();
```

See also

Qore::zzz8listzzz9::sizep()

45.63.2.10 bool Qore::zzz8listzzz9::sizep ( )

Returns True since lists can return a non-zero size.

Code Flags:

CONSTANT

Returns

True since lists can return a non-zero size

See also

Qore::zzz8listzzz9::size()

Since

Qore 0.8.9

45.63.2.11 int Qore::zzz8listzzz9::typeCode ( )

Returns Qore::NT_LIST.

Returns

Qore::NT_LIST

Code Flags:

CONSTANT

Example:

```
switch ($l.typeCode()) {
    case NT_LIST:
        printf("%y: is a list\n", $l);
        break;
}
```

45.63.2.12 bool Qore::zzz8listzzz9::val ( )

Returns False if the list is empty (size = 0), True if not.

The opposite of Qore::zzz8listzzz9::empty()

Returns

False if the list is empty (size = 0), True if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $l.val();
```

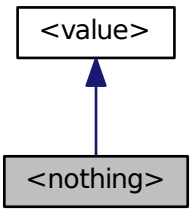
See also

- %perl-bool-eval
- %strict-bool-eval

45.64 Qore::zzz8nothingzzz9 Class Reference

Methods in this pseudo-class can be executed on NOTHING.

Inheritance diagram for Qore::zzz8nothingzzz9:



Public Member Functions

- SingleValueIterator contextIterator ()

  Returns an empty SingleValueIterator object.
- nothing firstKey ()

  Returns NOTHING.
- nothing firstValue ()

  Returns NOTHING.
- bool hasKey (softstring key)

  Returns True if the key exists in the hash (may or may not be assigned a value), False if not.
- bool hasKeyValue (softstring key)

  Returns True if the key exists and is assigned to a value, False if not.
- SingleValueIterator keyIterator ()

  Returns an empty SingleValueIterator object.
- list keys ()

  Returns an empty list.
- nothing lastKey ()

  Returns NOTHING.
- nothing lastValue ()

  Returns NOTHING.
- int lsize ()

  Returns 0.
- SingleValueIterator pairIterator ()

  Returns an empty SingleValueIterator object.
- SingleValueIterator rangeIterator ()

  Returns an empty SingleValueIterator object.
- int typeCode ()

  Returns Qore::NT_NOTHING.
- list values ()

  Returns an empty list.

45.64.1 Detailed Description

Methods in this pseudo-class can be executed on NOTHING.

45.64.2 Member Function Documentation

45.64.2.1 SingleValueIterator Qore::zzz8nothingzzz9::contextIterator ( )

Returns an empty SingleValueIterator object.

Returns

an empty SingleValueIterator object

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), get_hash_or_nothing().contextIterator();
```

See also

Qore::zzz8hashzzz9::contextIterator()

Since

Qore 0.8.6.2

45.64.2.2 nothing Qore::zzz8nothingzzz9::firstKey ( )

Returns NOTHING.

Returns

NOTHING

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my *string $n = $h.firstKey();
```

See also

Qore::zzz8hashzzz9::firstKey()

Since

Qore 0.8.7

45.64.2.3 nothing Qore::zzz8nothingzzz9::firstValue ( )

Returns NOTHING.

Returns

NOTHING

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my any $n = $h.firstValue();
```

See also

Qore::zzz8hashzzz9::firstValue()

Since

Qore 0.8.7

45.64.2.4 bool Qore::zzz8nothingzzz9::hasKey (softstring *key*)

Returns True if the key exists in the hash (may or may not be assigned a value), False if not.

Code Flags:

CONSTANT

Parameters

key	the key name to check

Returns

True if the key exists in the hash (may or may not be assigned a value), False if not

Example:

```
my *hash $h = hash_or_nothing();
my bool $b = $h.hasKey($key);
```

See also

Qore::zzz8hashzzz9::hasKey(softstring)

Since

Qore 0.8.7

45.64.2.5 bool Qore::zzz8nothingzzz9::hasKeyValue (softstring *key*)

Returns True if the key exists and is assigned to a value, False if not.

Code Flags:

CONSTANT

Parameters

key	the key name to check

Returns

True if the key exists and is assigned to a value, False if not

Example:

```
my *hash $h = hash_or_nothing();
my bool $b = $h.hasKeyValue($key);
```

See also

Qore::zzz8hashzzz9::hasKeyValue(softstring)

Since

Qore 0.8.7

45.64.2.6 SingleValueIterator Qore::zzz8nothingzzz9::keyIterator ( )

Returns an empty SingleValueIterator object.

Returns

an empty SingleValueIterator object

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), get_hash_or_nothing().keyIterator();
```

See also

Qore::zzz8hashzzz9::keyIterator()

Since

Qore 0.8.6.2

45.64.2.7 list Qore::zzz8nothingzzz9::keys ( )

Returns an empty list.

Returns

an empty list

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my list $l = $h.keys();
```

See also

Qore::zzz8hashzzz9::keys()

Since

Qore 0.8.7

45.64.2.8 nothing Qore::zzz8nothingzzz9::lastKey ( )

Returns NOTHING.

Returns

NOTHING

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my *string $n = $h.firstKey();
```

See also

Qore::zzz8hashzzz9::firstKey()

Since

Qore 0.8.7

45.64.2.9 nothing Qore::zzz8nothingzzz9::lastValue ( )

Returns NOTHING.

Returns

NOTHING

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my any $n = $h.firstValue();
```

See also

Qore::zzz8hashzzz9::lastValue()

Since

Qore 0.8.7

45.64.2.10 int Qore::zzz8nothingzzz9::lsize ( )

Returns 0.

Returns

0

Code Flags:

CONSTANT

Example:

```
printf("iterating %d element%s\n", $val.lsize(), $val.lsize() == 1 ? "" : "s");
foreach my any $element in ($val) {
    do_something($element);
}
```

45.64.2.11 SingleValueIterator Qore::zzz8nothingzzz9::pairIterator ( )

Returns an empty SingleValueIterator object.

Returns

an empty SingleValueIterator object

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), get_hash_or_nothing().pairIterator();
```

Since

Qore 0.8.6.2

45.64.2.12 SingleValueIterator Qore::zzz8nothingzzz9::rangeIterator ( )

Returns an empty SingleValueIterator object.

Returns

an empty SingleValueIterator object

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), get_list_or_nothing().rangeIterator();
```

See also

Qore::zzz8listzzz9::rangeIterator()

Since

Qore 0.8.8

45.64.2.13 int Qore::zzz8nothingzzz9::typeCode ( )

Returns Qore::NT_NOTHING.

Returns

Qore::NT_NOTHING

Code Flags:

CONSTANT

Example:

```
switch ($n.typeCode()) {
    case NT_NOTHING:
        printf("%y: no value\n", $n);
        break;
}
```

45.64.2.14 list Qore::zzz8nothingzzz9::values ( )

Returns an empty list.

Returns

an empty list

Code Flags:

CONSTANT

Example:

```
my *hash $h = hash_or_nothing();
my list $l = $h.values();
```

See also

Qore::zzz8hashzzz9::values()

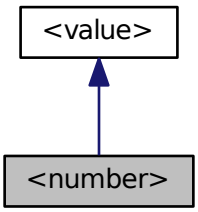
Since

Qore 0.8.7

45.65 Qore::zzz8numberzzz9 Class Reference

Methods in this pseudo-class can be executed on arbitrary precision number values.

Inheritance diagram for Qore::zzz8numberzzz9:



Public Member Functions

- number abs ()

  Returns the absolute value of the number.
- string format (string fmt)

  Returns a string of a formatted number according to a format string.
- bool infp ()

  Returns True if the number is infinity (+ or -)
- bool intp ()

  Returns True because number values can be converted to integers.
- bool nanp ()

  Returns True if the number is NaN (not a number)
- int prec ()

  Returns the precision of the current number.
- int sign ()

  Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.
- bool strp ()

  Returns True because number values can be converted to strings.
- string toString (int fmt)

  Returns the string representation of the number according to the format argument.
- int typeCode ()

  Returns Qore::NT_NUMBER.
- bool val ()

  Returns True if the number is non-zero, False if zero.

45.65.1 Detailed Description

Methods in this pseudo-class can be executed on arbitrary precision number values.

45.65.2 Member Function Documentation

45.65.2.1 number Qore::zzz8numberzzz9::abs ( )

Returns the absolute value of the number.

Code Flags:

CONSTANT

Example:

```
$n = $n.abs();
```

Returns

the absolute value of the number

Note

equivalent to abs(number)

Since

Qore 0.8.8

45.65.2.2 string Qore::zzz8numberzzz9::format (string *fmt*)

Returns a string of a formatted number according to a format string.

Code Flags:

CONSTANT

Parameters

fmt	the format string has the following format: <*thousands_separator*> [<*decimal_separator*><*decimals*>] where: • *thousands_separator* and *decimal_separator* are single ASCII characters defining the thousands and decimal separator characters respectively, and • *decimals* is a single digit defining how may decimals should appear after the decimal point

Returns

a string of a formatted number according to a format string; if the format string does not follow the given format, then an empty string is returned

Example:

```
my number $n = -48392093894.2349n;
my string $nstr = $n.format(".,3"); # returns "-48.392.093.894,235"
```

See also

- Qore::zzz8floatzzz9::format(string)
- Qore::zzz8intzzz9::format(string)

Since

Qore 0.8.6

45.65.2.3 bool Qore::zzz8numberzzz9::infp ( )

Returns True if the number is infinity (+ or -)

Returns

True if the number is infinity (+ or -)

Code Flags:

CONSTANT

Example:

```
if ($n.infp())
    print("the operation resulted in infinity\n");
```

Since

Qore 0.8.6

45.65.2.4 bool Qore::zzz8numberzzz9::intp ( )

Returns True because number values can be converted to integers.

Returns

True because number values can be converted to integers

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

Since

Qore 0.8.6

45.65.2.5 bool Qore::zzz8numberzzz9::nanp ( )

Returns True if the number is NaN (not a number)

Returns

True if the number is NaN (not a number)

Code Flags:

CONSTANT

Example:

```
if ($n.nanp())
    print("the operation resulted in NaN\n");
```

Since

Qore 0.8.6

45.65.2.6 int Qore::zzz8numberzzz9::prec ( )

Returns the precision of the current number.

Returns

the precision of the current number

Code Flags:

CONSTANT

Example:

```
printf("precision: %d\n", $n.prec());
```

Since

Qore 0.8.6

45.65.2.7 int Qore::zzz8numberzzz9::sign ( )

Returns -1 if the number is negative, 0 if it is zero, or 1 if it is positive.

Returns

-1 if the number is negative, 0 if it is zero, or 1 if it is positive

Code Flags:

CONSTANT

Example:

```
printf("sign: %d\n", $n.sign());
```

Since

Qore 0.8.6

45.65.2.8 bool Qore::zzz8numberzzz9::strp ( )

Returns True because number values can be converted to strings.

Returns

True because number values can be converted to strings

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

Since

Qore 0.8.6

45.65.2.9 string Qore::zzz8numberzzz9::toString (int *fmt*)

Returns the string representation of the number according to the format argument.

Code Flags:

CONSTANT

Parameters

fmt	see Number Formatting Constants for possible values; note that this argument is interpreted as a bit field; if the format argument is not a valid format value (i.e. if none of the formatting bits are set in the argument) then NF_Default is assumed instead

Returns

the string representation of the number according to the format argument

Since

Qore 0.8.6

45.65.2.10 int Qore::zzz8numberzzz9::typeCode ( )

Returns Qore::NT_NUMBER.

Returns

Qore::NT_NUMBER

Code Flags:

CONSTANT

Example:

```
switch ($f.typeCode()) {
    case NT_NUMBER:
        printf("%y: is a number\n", $f);
        break;
}
```

Since

Qore 0.8.6

45.65.2.11 bool Qore::zzz8numberzzz9::val ( )

Returns True if the number is non-zero, False if zero.

Returns

True if the number is non-zero, False if zero

Code Flags:

CONSTANT

Example:

```
my bool $b = $f.val();
```

See also

- %perl-bool-eval
- %strict-bool-eval

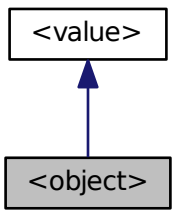
Since

Qore 0.8.6

45.66 Qore::zzz8objectzzz9 Class Reference

Methods in this pseudo-class can be executed on objects.

Inheritance diagram for Qore::zzz8objectzzz9:



Public Member Functions

- string className ()

  Returns the class name of the object.
- bool empty ()

  Returns True if the object has no public or private members, False if it does.

- __7_ string firstKey ()

  Returns the first member name in the object or NOTHING if the object has no members; if called from outside the object, the first public member name is returned (if any)
- bool hasCallableMethod (string name)

  Returns True if the given method exists (can be non-static or static) and is callable from the current context.
- bool hasCallableNormalMethod (string name)

  Returns True if the given non-static method exists and is callable from the current context.
- bool hasCallableStaticMethod (string name)

  Returns True if the given static method exists and is callable from the current context.
- bool isSystem ()

  Returns True if the object is a system object (ie a constant object like stdin, etc), False if not.
- AbstractIterator iterator ()

  Returns an ObjectIterator object for the hash.
- ObjectKeyIterator keyIterator ()

  Returns a ObjectKeyIterator object for the hash.
- list keys ()

  Returns a list of member names of the object; if called from outside the object, only public members are returned.
- __7_ string lastKey ()

  Returns the last member name in the object or NOTHING if the object has no members; if called from outside the object, the last public member name is returned (if any)
- ObjectPairIterator pairIterator ()

  Returns a ObjectPairIterator object for the hash.
- int size ()

  Returns the number of members in the object, public and private.
- bool sizep ()

  Returns True since objects can return a non-zero size.
- int typeCode ()

  Returns Qore::NT_OBJECT.
- bool val ()

  Returns False if the object has no public or private members, True if it does.

45.66.1 Detailed Description

Methods in this pseudo-class can be executed on objects.

45.66.2 Member Function Documentation

45.66.2.1 string Qore::zzz8objectzzz9::className ( )

Returns the class name of the object.

Returns

the class name of the object

Code Flags:

CONSTANT

Example:

```
my string $cn = $o.className();
```

45.66.2.2 bool Qore::zzz8objectzzz9::empty ( )

Returns True if the object has no public or private members, False if it does.

The opposite of Qore::zzz8objectzzz9::val()

Returns

True if the object has no public or private members, False if it does

Code Flags:

RET_VALUE_ONLY

Example:

```
my bool $b = $o.empty();
```

45.66.2.3 __7_ string Qore::zzz8objectzzz9::firstKey ( )

Returns the first member name in the object or NOTHING if the object has no members; if called from outside the object, the first public member name is returned (if any)

Returns

the first member name in the object or NOTHING if the object has no members; if called from outside the object, the first public member name is returned (if any)

Code Flags:

RET_VALUE_ONLY

Example:

```
my *string $n = $o.firstKey();
```

See also

Qore::zzz8objectzzz9::lastKey()

45.66.2.4 bool Qore::zzz8objectzzz9::hasCallableMethod (string *name*)

Returns True if the given method exists (can be non-static or static) and is callable from the current context.

Code Flags:

CONSTANT

Example:

```
my bool $b = $obj.hasCallableMethod("getStatus");
```

Returns

True if the given method exists (can be non-static or static) and is callable from the current context

See also

- hasCallableNormalMethod()
- hasCallableStaticMethod

Note

- returns False if the method doesn't exist or is not callable due to being private, for example
- if a methodGate() method exists but no explicit definition of the given method exists, then this pseudo-method will return False, but any method name can be called on the object due to the existence of the methodGate() method
- this pseudo-method will also return True if the given method exists and is accessible in an inherited class

Since

Qore 0.8.8

45.66.2.5 bool Qore::zzz8objectzzz9::hasCallableNormalMethod (string *name*)

Returns True if the given non-static method exists and is callable from the current context.

Code Flags:

CONSTANT

Example:

```
my bool $b = $obj.hasCallableNormalMethod("getStatus");
```

Returns

True if the given non-static method exists and is callable from the current context

See also

- hasCallableMethod()
- hasCallableStaticMethod

Note

- returns False if the method doesn't exist or is not callable due to being private, for example
- if a methodGate() method exists but no explicit definition of the given method exists, then this pseudo-method will return False, but any method name can be called on the object due to the existence of the methodGate() method
- this pseudo-method will also return True if the given method exists and is accessible in an inherited class

Since

Qore 0.8.8

45.66.2.6 bool Qore::zzz8objectzzz9::hasCallableStaticMethod (string *name*)

Returns True if the given static method exists and is callable from the current context.

Code Flags:

CONSTANT

Example:

```
my bool $b = $obj.hasCallableStaticMethod("getStatus");
```

Returns

True if the given static method exists and is callable from the current context

See also

- hasCallableMethod()
- hasCallableStaticMethod

Note

- returns False if the static method doesn't exist or is not callable due to being private, for example
- this pseudo-method will also return True if the given static method exists and is accessible in an inherited class

Since

Qore 0.8.8

45.66.2.7 bool Qore::zzz8objectzzz9::isSystem ( )

Returns True if the object is a system object (ie a constant object like stdin, etc), False if not.

Returns

True if the object is a system object (ie a constant object like stdin, etc), False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $i.isSystem();
```

45.66.2.8 AbstractIterator Qore::zzz8objectzzz9::iterator ( )

Returns an ObjectIterator object for the hash.

Returns

an ObjectIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), $obj.iterator();
```

Since

Qore 0.8.6

45.66.2.9 ObjectKeyIterator Qore::zzz8objectzzz9::keyIterator ( )

Returns a ObjectKeyIterator object for the hash.

Returns

a ObjectKeyIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %s\n", $1), $obj.keyIterator();
```

Since

Qore 0.8.6.2

45.66.2.10 list Qore::zzz8objectzzz9::keys ( )

Returns a list of member names of the object; if called from outside the object, only public members are returned.

Returns

a list of member names of the object; if called from outside the object, only public members are returned

Code Flags:

RET_VALUE_ONLY

Example:

```
my list $l = $o.keys();
```

45.66.2.11 __7_ string Qore::zzz8objectzzz9::lastKey ( )

Returns the last member name in the object or NOTHING if the object has no members; if called from outside the object, the last public member name is returned (if any)

Returns

the last member name in the object or NOTHING if the object has no members; if called from outside the object, the last public member name is returned (if any)

Code Flags:

RET_VALUE_ONLY

Example:

```
my *string $n = $o.lastKey();
```

See also

Qore::zzz8objectzzz9::firstKey()

45.66.2.12 ObjectPairIterator Qore::zzz8objectzzz9::pairIterator ( )

Returns a ObjectPairIterator object for the hash.

Returns

a ObjectPairIterator object for the hash

Code Flags:

CONSTANT

Example:

```
map printf("+ %s\n", $1), $obj.pairIterator();
```

Since

Qore 0.8.6.2

45.66.2.13 int Qore::zzz8objectzzz9::size ( )

Returns the number of members in the object, public and private.

Returns

the number of members in the object, public and private

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $num = $o.size();
```

See also

Qore::zzz8objectzzz9::sizep()

45.66.2.14 bool Qore::zzz8objectzzz9::sizep ( )

Returns True since objects can return a non-zero size.

Code Flags:

CONSTANT

Returns

True since objects can return a non-zero size

See also

Qore::zzz8objectzzz9::size()

Since

Qore 0.8.9

45.66.2.15 int Qore::zzz8objectzzz9::typeCode ( )

Returns Qore::NT_OBJECT.

Returns

Qore::NT_OBJECT

Code Flags:

CONSTANT

Example:

```
switch ($o.typeCode()) {
    case Qore::NT_OBJECT:
        printf("%y: is an object\n", $o);
        break;
}
```

45.66.2.16 bool Qore::zzz8objectzzz9::val ( )

Returns False if the object has no public or private members, True if it does.

The opposite of Qore::zzz8objectzzz9::empty()

Returns

False if the object has no public or private members, True if it does

Code Flags:

RET_VALUE_ONLY

Example:

```
my bool $b = $o.val();
```

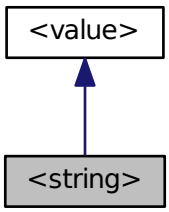
See also

- %perl-bool-eval
- %strict-bool-eval

45.67 Qore::zzz8stringzzz9 Class Reference

Methods in this pseudo-class can be executed on strings.

Inheritance diagram for Qore::zzz8stringzzz9:



Public Member Functions

- int comparePartial (string ostr)

  Compares the beginning of the current string with a shorter string passed as an argument, returns -1, 0, or 1 if the argument string is less than, equal, or greater than the beginning of the current string; returns also -1 if the argument string is equal to the beginning of the current string but the argument string is longer than the current string.
- bool empty ()

  Returns True if the string is empty, False if not.
- string encoding ()

  Returns the name of the string's character encoding.
- bool equalPartial (string ostr)

  Compares the beginning of the current string with a shorter string passed as an argument for equality only, returns True if the string argument matches the beginning of the string, False if not.
- bool equalPartialPath (string ostr)

  Compares the beginning of the current string assumed to be a path with a shorter string passed as an argument for equality only, returns True if the string argument matches the beginning of the string where either both strings are the same size or the current string has a '/' or '?' character after the point where the argument string stops, False if not.
- int find (softstring substr, softint pos=0)

  Retrieves the character position of a substring within a string.
- __7_ string getLine (int offset=0, __7_ string eol, bool trim=True, __7_ reference size)

  returns a string for the next line in the string buffer starting at the given offset (or at the beginning if no offset is given)
- int getUnicode (int offset=0)

  returns the Unicode code for the given character offset in the string
- bool intp ()

  Returns True if the string can be converted to an integer, False if not, this depends on the first (or possibly second) character of the string, if it's 0 - 9 (possibly preceded by "-"), then the method returns True.
- bool isDataAscii ()

  returns True if the string is empty or has no characters with the high bit set (ie all characters < 128)
- bool isDataPrintableAscii ()

  returns True if the string is empty or only contains printable non-control ASCII characters (ie all characters > 31 && < 127)
- int length ()

  Returns the number of characters in the string; may not be equal to the byte length (returned by Qore::zzz8stringzzz9::strlen() and Qore::zzz8stringzzz9::size()) for multi-byte character encodings.

- string lwr ()

  Returns the string in lower case.
- bool regex (string regex, int options=0)

  Returns True if the regular expression matches the string passed, otherwise returns False.
- __7_ list regexExtract (string regex, int options=0)

  Returns a list of substrings in a string based on matching patterns defined by a regular expression.
- int rfind (softstring substr, softint pos=-1)

  Retrieves the character position of a substring within a string, starting the search from the end of the string.
- int size ()

  Returns the number of bytes in the string (not including the terminating null character ('\0')
- bool sizep ()

  Returns True since strings can return a non-zero size.
- list split (string sep, bool with_separator=False)

  Splits a string into a list of components based on a separator string.
- list split (string sep, string quote, bool trim_unquoted=False)

  Splits a string into a list of components based on a separator string and a quote character.
- int strlen ()

  Returns the number of bytes in the string (not including the terminating null character ('\0')
- bool strp ()

  Returns True by default.
- string substr (softint start)

  Returns a portion of a string starting from an integer offset.
- string substr (softint start, softint len)

  Returns a portion of a string starting from an integer offset, with a length parameter.
- string toBase64 (softint maxlinelen=-1)

  Returns the base64-encoded representation of the string.
- string toHex ()

  returns a string of hexadecimal digits corresponding to the contents of the string
- string toMD5 ()

  Returns the MD5 message digest of the string as a hex string.
- string toSHA1 ()

  Returns the SHA1 message digest of the string as a hex string.
- string toSHA224 ()

  Returns the SHA-224 message digest (a variant of SHA-2) of the string as a hex string.
- string toSHA256 ()

  Returns the SHA-256 message digest (a variant of SHA-2) of the string as a hex string.
- string toSHA384 ()

  Returns the SHA-384 message digest (a variant of SHA-2) of the string as a hex string.
- string toSHA512 ()

  Returns the SHA-512 message digest (a variant of SHA-2) of the string as a hex string.
- int typeCode ()

  Returns Qore::NT_STRING.
- string unaccent ()

  Returns a string with all accented characters removed.
- string upr ()

  Returns the string in upper case.
- bool val ()

  Returns False if the string is empty, True if not.

45.67.1 Detailed Description

Methods in this pseudo-class can be executed on strings.

45.67.2 Member Function Documentation

45.67.2.1 int Qore::zzz8stringzzz9::comparePartial (string *ostr*)

Compares the beginning of the current string with a shorter string passed as an argument, returns -1, 0, or 1 if the argument string is less than, equal, or greater than the beginning of the current string; returns also -1 if the argument string is equal to the beginning of the current string but the argument string is longer than the current string.

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $i = $str.comparePartial($ostr);
```

Parameters

ostr	the partial string to compare the current string to

Returns

-1, 0, or 1 if the argument string is less than, equal, or greater than the beginning of the current string; returns also -1 if the argument string is equal to the beginning of the current string but the argument string is longer than the current string

Since

Qore 0.8.8

45.67.2.2 bool Qore::zzz8stringzzz9::empty ( )

Returns True if the string is empty, False if not.

The opposite of Qore::zzz8stringzzz9::val()

Returns

True if the string is empty, False if not

Code Flags:

CONSTANT

Example:

```
my bool $b = $str.empty();
```

45.67.2.3 string Qore::zzz8stringzzz9::encoding ( )

Returns the name of the string's character encoding.

Returns

the name of the string's character encoding

Code Flags:

CONSTANT

Example:

```
my string $enc = $str.encoding();
```

Note

equivalent to Qore::get_encoding(string)

45.67.2.4 bool Qore::zzz8stringzzz9::equalPartial (string *ostr*)

Compares the beginning of the current string with a shorter string passed as an argument for equality only, returns True if the string argument matches the beginning of the string, False if not.

Code Flags:

RET_VALUE_ONLY

Example:

```
my bool $b = $str.equalPartial($ostr);
```

This pseudo-method is slightly faster than comparePartial() since the length of the substring can be used to determine if the strings can match or not.

Parameters

ostr	the partial string to compare the current string to

Returns

True if the string argument matches the beginning of the string, False if not

Since

Qore 0.8.8

45.67.2.5 bool Qore::zzz8stringzzz9::equalPartialPath (string *ostr*)

Compares the beginning of the current string assumed to be a path with a shorter string passed as an argument for equality only, returns True if the string argument matches the beginning of the string where either both strings are the same size or the current string has a '/' or '?' character after the point where the argument string stops, False if not.

Code Flags:

RET_VALUE_ONLY

Example:

```
my bool $b = $str.equalPartialPath($ostr);
```

Parameters

ostr	the partial string to compare the current string to

Returns

True if the string argument matches the beginning of the string where either both strings are the same size or the current string has a '/' or '?' character after the point where the argument string stops, False if not

Since

Qore 0.8.8

45.67.2.6 int Qore::zzz8stringzzz9::find (softstring *substr*, softint *pos* = 0)

Retrieves the character position of a substring within a string.

The *pos* argument and the return value are in character positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

RET_VALUE_ONLY

Parameters

substr	the substring to find in the string; if the character encoding of this string does not match *str*, then it will be converted to *str*'s character encoding before processing
pos	the starting character position for the search

Returns

the character position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = $str.find($substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::index(softstring, softstring, softint)

See also

- Qore::zzz8stringzzz9::rfind(softstring, softint)
- rindex(softstring, softstring, softint)
- bindex(softstring, softstring, softint)
- brindex(softstring, softstring, softint)

45.67.2.7 __7_ string Qore::zzz8stringzzz9::getLine (int *offset* = 0, __7_ string *eol*, bool *trim* = True, __7_ reference *size*)

returns a string for the next line in the string buffer starting at the given offset (or at the beginning if no offset is given)

Code Flags:

CONSTANT

Example:

```
my *string $line = $string.getLine($pos);
```

Parameters

offset	the offset in bytes from the beginning of the string; negative numbers give an offset from the end of the string
eol	the optional end of line character(s) to use to detect lines in the buffer; if this string is not passed, then the end of line character(s) are detected automatically, and can be either `"\n"`, `"\r"`, or `"\r\n"`; if this string is passed and has a different character encoding from this object's (as determined by the `encoding` parameter), then it will be converted to the string's character encoding
trim	if True the string return values for the lines iterated will be trimmed of the eol bytes
size	an optional reference to an integer that returns the number of bytes in the line including the end of line characters

Returns

the line found or NOTHING if no end of line character(s) were found or no data was found after the offset

Since

Qore 0.8.9

45.67.2.8 int Qore::zzz8stringzzz9::getUnicode (int *offset* = 0)

returns the Unicode code for the given character offset in the string

Code Flags:

RET_VALUE_ONLY

Example:

```
my int $uc = $str.getUnicode(0);
```

Parameters

offset	the offset in characters in the string; negative numbers give offsets from the end of the string

Returns

the Unicode code for the given character offset in the string

Since

Qore 0.8.8

45.67.2.9 bool Qore::zzz8stringzzz9::intp ( )

Returns True if the string can be converted to an integer, False if not, this depends on the first (or possibly second) character of the string, if it's 0 - 9 (possibly preceded by "-"), then the method returns True.

Returns

True if the string can be converted to an integer, False if not, this depends on the first (or possibly second) character of the string, if it's 0 - 9 (possibly preceded by "-"), then the method returns True

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

45.67.2.10 bool Qore::zzz8stringzzz9::isDataAscii ( )

returns True if the string is empty or has no characters with the high bit set (ie all characters < 128)

Returns

True if the string is empty or has no characters with the high bit set (ie all characters < 128)

Code Flags:

CONSTANT

Example:

```
my bool $b = $str.isDataAscii();
```

Since

Qore 0.8.6

45.67.2.11 bool Qore::zzz8stringzzz9::isDataPrintableAscii ( )

returns True if the string is empty or only contains printable non-control ASCII characters (ie all characters > 31 && < 127)

Returns

True if the string is empty or only contains printable non-control ASCII characters (ie all characters > 31 && < 127)

Code Flags:

CONSTANT

Example:

```
my bool $b = $str.isDataPrintableAscii();
```

Since

Qore 0.8.6

45.67.2.12 int Qore::zzz8stringzzz9::length ( )

Returns the number of characters in the string; may not be equal to the byte length (returned by Qore::zzz8stringzzz9::strlen() and Qore::zzz8stringzzz9::size()) for multi-byte character encodings.

Returns

the number of characters in the string; may not be equal to the byte length (returned by Qore::zzz8stringzzz9::strlen() and Qore::zzz8stringzzz9::size()) for multi-byte character encodings

Code Flags:

CONSTANT

Example:

```
my int $len = $str.length();
```

Note

- this operation has *O(n)* complexity if the string has a multi-byte character encoding, otherwise it is *O(1)*
- equivalent to Qore::length(softstring)

See also

- Qore::zzz8stringzzz9::strlen()
- Qore::zzz8stringzzz9::size()

45.67.2.13 string Qore::zzz8stringzzz9::lwr ( )

Returns the string in lower case.

Code Flags:

CONSTANT

Example:

```
printf("%y", "PŘÍLIŠ ŽLUŤOUČKÝ KŮŇ ÚPĚL ĎÁBELSKÉ ÓDY".lwr());
# outputs: "příliš žlut'oučký kůň úpěl d'ábelské ódy"
```

This pseudo-method operates on a very wide range of non-ASCII characters using a Unicode lookup table for mapping Latin, Cyrillic, Greek, Armenian, Georgian, etc characters.

Returns

the string in lower case

Note

- equivalent to Qore::tolower(string)

See also

- Qore::zzz8stringzzz9::upr()
- tolower()
- toupper()

Since

Qore 0.8.8 this pseudo-method operates on a wide range of characters and is no longer limited to ASCII characters

45.67.2.14 bool Qore::zzz8stringzzz9::regex (string *regex*, int *options* = 0)

Returns True if the regular expression matches the string passed, otherwise returns False.

Strings are converted to UTF-8 for pattern-matching; if any invalid encodings are encountered, an ENCODING-C↩ ONVERSION-ERROR is raised

Code Flags:

RET_VALUE_ONLY

Parameters

regex	the regular expression pattern
options	regular expression options; see Regular Expression Constants for possible values

Returns

True if the regular expression matches the string passed, otherwise returns False

Example:

```
my bool $b = "hello".regex("^hel"); # returns True
```

Exceptions

REGEX-COMPILATION-ERROR	There was an error compiling the regular expression
REGEX-OPTION-ERROR	the option argument contains invalid option bits
ENCODING-CONVERSION-ERROR	this exception could be thrown if an encoding error is encountered when converting the given strings to UTF-8

Note

equivalent to Qore::regex(string, string, int)

See also

Regular Expressions for more information about regular expression support in Qore

Since

Qore 0.8.5

45.67.2.15 __7_ list Qore::zzz8stringzzz9::regexExtract (string *regex*, int *options* = 0)

Returns a list of substrings in a string based on matching patterns defined by a regular expression.

Strings are converted to UTF-8 for pattern-matching; if any invalid encodings are encountered, an ENCODING-C↩ ONVERSION-ERROR is raised

Code Flags:

RET_VALUE_ONLY

Parameters

regex	the regular expression to use for matching, elements should be given in parentheses
options	regular expression options; see Regular Expression Constants for possible values

Returns

a list of substrings in a string based on matching patterns defined by a regular expression or NOTHING if no match was made

Example:

```
my string $str = "ns:element";
my *list $rv = $str.regexExtract("(\\w+):(\\w+)");
```

Exceptions

REGEX-COMPILATION-ERROR	There was an error compiling the regular expression
REGEX-OPTION-ERROR	the option argument contains invalid option bits
ENCODING-CONVERSION-ERROR	this exception could be thrown if an encoding error is encountered when converting the given strings to UTF-8

Note

equivalent Qore::regex_extract(string, string, int)

See also

Regular Expressions for more information about regular expression support in Qore

Since

Qore 0.8.5
Qore 0.8.8 this function accepts the Qore::RE_Global option to extract all occurrences of the pattern(s) in a string

45.67.2.16 int Qore::zzz8stringzzz9::rfind (softstring *substr*, softint *pos* = -1)

Retrieves the character position of a substring within a string, starting the search from the end of the string.

The *pos* argument and the return value are in character positions; byte offsets may differ from the character offsets with multi-byte character encodings.

Code Flags:

RET_VALUE_ONLY

Parameters

substr	the substring to find in *str*; if the character encoding of this string does not match *str*, then it will be converted to *str*'s character encoding before processing
pos	the starting character position for the search, -1 means start from the end of the string

Returns

the character position of a substring within a string, -1 is returned if the substring is not found

Example:

```
my int $i = $str.rfind($substr);
if ($i == -1)
    printf("could not find %y in %y\n", $substr, $str);
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::rindex(softstring, softstring, softint)

See also

- Qore::zzz8stringzzz9::find(softstring, softint)
- index(softstring, softstring, softint)
- bindex(softstring, softstring, softint)
- brindex(softstring, softstring, softint)

45.67.2.17 int Qore::zzz8stringzzz9::size ( )

Returns the number of bytes in the string (not including the terminating null character ('\0')

Returns

the number of bytes in the string (not including the terminating null character ('\0')

Code Flags:

CONSTANT

Example:

```
my int $len = $str.size();
```

Note

- this operation is always executed in constant time (ie *O(1)*) because the string's byte length is always stored with the string
- equivalent to Qore::zzz8stringzzz9::strlen() and Qore::strlen(softstring)

See also

- Qore::zzz8stringzzz9::sizep()
- Qore::zzz8stringzzz9::length()

45.67.2.18 bool Qore::zzz8stringzzz9::sizep ( )

Returns True since strings can return a non-zero size.

Code Flags:

CONSTANT

Returns

True since strings can return a non-zero size

See also

Qore::zzz8stringzzz9::size()

Since

Qore 0.8.9

45.67.2.19 list Qore::zzz8stringzzz9::split (string *sep*, bool *with_separator* = False)

Splits a string into a list of components based on a separator string.

Code Flags:

RET_VALUE_ONLY

Parameters

sep	the separator string; if the separator string is not found in the string to split, then a list with only one element containing the entire string argument is returned'; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
with_separator	include the separator string in every element

Returns

a list of each component of a string separated by a separator string, with the separator removed; the separator pattern will not be included in the elements of the list returned unless the *with_separator* argument is True

Example:

```
my string $str = "some:text:here";
my list $list = $str.split(":"); # returns ("some", "text", "here")
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion

Note

equivalent to Qore::split(string, string, bool)

Since

Qore 0.8.5

45.67.2.20 list Qore::zzz8stringzzz9::split (string *sep*, string *quote*, bool *trim_unquoted* = False)

Splits a string into a list of components based on a separator string and a quote character.

The quote character can appear as the first part of a field, in which case it is assumed to designate the entire field. If instances of the quote character are found in the field preceded by a backquote character ("\"), then these quote characters are included as part of the field's text and not treated as quote characters. Also the separator character can appear as a part of a field with this variant. This variant is useful for parsing CSV files, for example.

Code Flags:

RET_VALUE_ONLY

Parameters

sep	the separator string; if the separator string is not found in the string to split, then a list with only one element containing the entire string argument is returned'; if this string has a different character encoding than *str*, then it will be converted to *str*'s character encoding
quote	the quote character
trim_unquoted	remove leading and trailing whitespace from unquoted fields

Returns

a list of each component of a string separated by a separator string, with the separator and any enclosing quote characters removed

Example:

```
my list $list = "some,'text with spaces, and commas, here is another one! ,',here".split(",", "'"); #
      returns ("some", ", and commas, here is another one! ,", "here")
```

Exceptions

ENCODING-CONVERSION-ERROR	this exception could be thrown if the string arguments have different character encodings and an error occurs during encoding conversion
SPLIT-ERROR	field missing closing quote character; extra text following quoted field

Note

equivalent to Qore::split(string, string, string, bool)

Since

- Qore 0.8.5 added this pseudo-method
- Qore 0.8.6 added the `trim_unquoted` parameter

45.67.2.21 int Qore::zzz8stringzzz9::strlen ( )

Returns the number of bytes in the string (not including the terminating null character (`'\0'`)

Returns

the number of bytes in the string (not including the terminating null character (`'\0'`)

Code Flags:

CONSTANT

Example:

```
my int $len = $str.strlen();
```

Note

- this operation is always executed in constant time (ie *O(1)*) because the string's byte length is always stored with the string
- equivalent to Qore::zzz8stringzzz9::size() and Qore::strlen(softstring)

See also

Qore::zzz8stringzzz9::length()

45.67.2.22 bool Qore::zzz8stringzzz9::strp ( )

Returns True by default.

Returns

True by default

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.67.2.23 string Qore::zzz8stringzzz9::substr (softint *start*)

Returns a portion of a string starting from an integer offset.

Arguments can be negative, giving offsets from the end of the string. All offsets are character positions, not byte positions.

Code Flags:

RET_VALUE_ONLY

Parameters

start	The starting character for the substring where the first character is at offset 0; if the offset is negative, it designates the number of characters from the end of the string. If the offset is 0, then the entire string is returned.

Returns

the substring of the string starting from an integer character offset; the rest of the string is returned after this offset; an emtpy string is returned if the argument cannot be satisfied

Example:

```
# get the last 10 characters of a string
my string $substr = $str.substr(-10);
```

Exceptions

INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::substr(softstring, softint)

45.67.2.24 string Qore::zzz8stringzzz9::substr (softint *start*, softint *len*)

Returns a portion of a string starting from an integer offset, with a length parameter.

Arguments can be negative, giving offsets from the end of the string. All offsets are character positions, not byte positions.

Code Flags:

RET_VALUE_ONLY

Parameters

start	The starting character for the substring where the first character is at offset 0; if the offset is negative, it designates the number of characters from the end of the string
len	The maximum number of characters to copy; if this value is negative, the rest of the string from *start* will be copied to the substring, except without - *len* characters from the end of the string

Returns

the substring of the string according to the arguments passed; an emtpy string is returned if the argument cannot be satisfied

Example:

```
# get a substring 10 characters into the string except omitting the last 2 characters of the string
my string $substr = $str.substr(10, -2);
```

Exceptions

INVALID-ENCODING	this exception could be thrown if a character offset calculation fails due to invalid encoding of multi-byte character data

Note

equivalent to Qore::substr(softstring, softint, softint)

Since

Qore 0.8.5

45.67.2.25 string Qore::zzz8stringzzz9::toBase64 (softint *maxlinelen* = -1)

Returns the base64-encoded representation of the string.

Code Flags:

CONSTANT

Example:

```
my string $base64 = $str.toBase64(64);
```

Implementation based on RFC-1421 and RFC-2045

Parameters

maxlinelen	the maximum length of a line in the resulting output string in bytes; if this value is > 0 then output lines will be separated by CRLF characters

Returns

the base64-encoded string of the data passed

Since

Qore 0.8.8

See also

- Qore::zzz8binaryzzz9::toBase64()
- makeBase64String(string, softint)

45.67.2.26 string Qore::zzz8stringzzz9::toHex ( )

returns a string of hexadecimal digits corresponding to the contents of the string

Code Flags:

CONSTANT

Example:

```
my string $str = $str.toHex();
```

Returns

a string of hexadecimal digits corresponding to the contents of the string

Since

Qore 0.8.8

See also

- Qore::zzz8binaryzzz9::toHex()
- makeHexString(string)

45.67.2.27 string Qore::zzz8stringzzz9::toMD5 ( )

Returns the `MD5 message digest` of the string as a hex string.

The trailing null character is not included in the digest returned.

Returns

a hex string of the digest (ex: `"5d41402abc4b2a76b9719d911017c592"`)

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $str = "hello".toMD5(); # returns "5d41402abc4b2a76b9719d911017c592"
```

Exceptions

MD5-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

- equivalent to Qore::MD5()
- the MD5 algorithm is not collision-resistant; it's recommended to use another hash algorithm (like SHA-256) if cryptographic security is important

See also

- MD5_bin()
- Qore::zzz8stringzzz9::toMD5()

Since

Qore 0.8.5

45.67.2.28 string Qore::zzz8stringzzz9::toSHA1 ( )

Returns the SHA1 message digest of the string as a hex string.

The trailing null character is not included in the digest returned.

Returns

a hex string of the digest (ex: "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d")

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $str = "hello".toSHA1(); # returns "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

Exceptions

SHA1-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::SHA1()

See also

- SHA1_bin()
- Qore::zzz8stringzzz9::toSHA1()

Since

Qore 0.8.5

45.67.2.29 string Qore::zzz8stringzzz9::toSHA224 ( )

Returns the SHA-224 message digest (a variant of SHA-2) of the string as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA224

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193")

Example:

```
my string $str = "hello".toSHA224("hello"); # returns "
      ea09ae9cc6768c50fcee903ed054556e5bfc8347907f12598aa24193"
```

Exceptions

SHA224-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::SHA224()

See also

- SHA224_bin()
- Qore::zzz8stringzzz9::toSHA224()

Since

Qore 0.8.5

45.67.2.30 string Qore::zzz8stringzzz9::toSHA256 ( )

Returns the SHA-256 message digest (a variant of SHA-2) of the string as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA256

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Example:

```
my string $str = "hello".toSHA256(); # returns "
      2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
```

Exceptions

SHA256-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::SHA256()

See also

- SHA256_bin()
- Qore::zzz8stringzzz9::toSHA256()

Since

Qore 0.8.5

45.67.2.31 string Qore::zzz8stringzzz9::toSHA384 ( )

Returns the SHA-384 message digest (a variant of SHA-2) of the string as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA384

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90

Example:

```
my string $str = "hello".toSHA384(); # returns "
      59e1748777448c69de6b800d7a33bbfb9ff1b463e44354c3553bcdb9c666fa90125a3c79f90397bdf5f6a13de828684f"
```

Exceptions

SHA384-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::SHA384()

See also

- SHA384_bin()
- Qore::zzz8stringzzz9::toSHA224()

Since

Qore 0.8.5

45.67.2.32 string Qore::zzz8stringzzz9::toSHA512 ( )

Returns the SHA-512 message digest (a variant of SHA-2) of the string as a hex string.

The trailing null character is not included in the digest returned.

Platform Availability:

Qore::Option::HAVE_SHA512

Code Flags:

RET_VALUE_ONLY

Returns

a hex string of the digest (ex: "9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca7

Example:

```
my string $str = "hello".toSHA512(); # returns "
      9b71d224bd62f3785d96d46ad3ea3d73319bfbc2890caadae2dff72519673ca72323c3d99ba5c11d7c7acc6e14b8c5da0c4663475c2e5c3adef
```

Exceptions

SHA512-DIGEST-ERROR	error calculating digest (should not normally happen)

Note

equivalent to Qore::SHA512()

See also

- SHA512_bin()
- Qore::zzz8stringzzz9::toSHA512()

Since

Qore 0.8.5

45.67.2.33 int Qore::zzz8stringzzz9::typeCode ( )

Returns Qore::NT_STRING.

Returns

Qore::NT_STRING

Code Flags:

CONSTANT

Example:

```
switch ($str.typeCode()) {
    case NT_STRING:
        printf("%y: is a string\n", $str);
        break;
}
```

45.67.2.34 string Qore::zzz8stringzzz9::unaccent ( )

Returns a string with all accented characters removed.

Code Flags:

RET_VALUE_ONLY

Example:

```
my string $s = "příliš žlut'oučký kůň úpěl d'ábelské ódy";
printf("%y\n", $s.unaccent()); # result: "prilis zlutoucky kun upel dabelske ody"
```

The returned string has the same encoding as the original input.

Returns

the string with accents replaced

Since

Qore 0.8.8

45.67.2.35 string Qore::zzz8stringzzz9::upr ( )

Returns the string in upper case.

Code Flags:

CONSTANT

Example:

```
printf("%y", "příliš žlut'oučký kůň úpěl d'ábelské ódy".upr());
# outputs: "PŘÍLIŠ ŽLUŤOUČKÝ KŮŇ ÚPĚL ĎÁBELSKÉ ÓDY"
```

This pseudo-method operates on a very wide range of non-ASCII characters using a Unicode lookup table for mapping Latin, Cyrillic, Greek, Armenian, Georgian, etc characters.

Returns

the string in upper case

Note

- equivalent to Qore::toupper(string)

See also

- Qore::zzz8stringzzz9::lwr()
- tolower()
- toupper()

Since

Qore 0.8.8 this pseudo-method operates on a wide range of characters and is no longer limited to ASCII characters

45.67.2.36 bool Qore::zzz8stringzzz9::val ( )

Returns False if the string is empty, True if not.

Returns

False if the string is empty, True if not

Code Flags:

CONSTANT

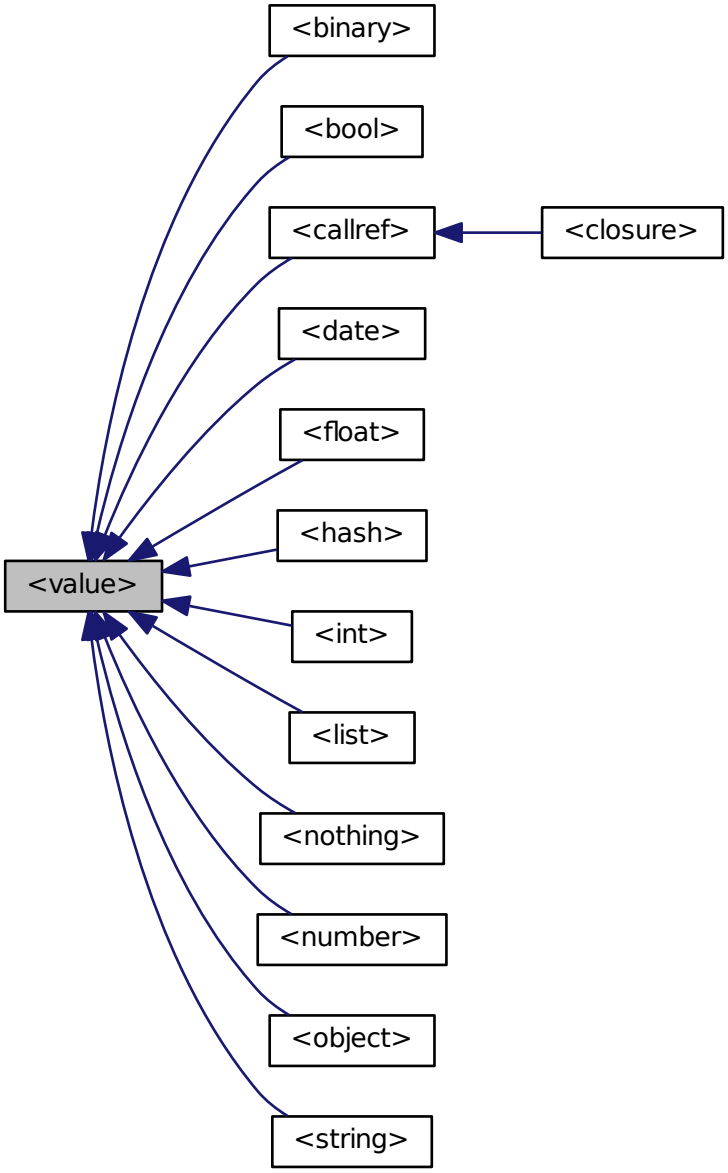
Example:

```
my bool $b = $str.val();
```

45.68 Qore::zzz8valuezzz9 Class Reference

Methods in this pseudo-class are available to be executed on any value type (even NOTHING); this is the root class for all pseudo-classes.

Inheritance diagram for Qore::zzz8valuezzz9:



Public Member Functions

- bool callp ()

  Returns False; this method is reimplemented in other types and will return True if the given expression is a callable value (ie closures or call references)
- bool empty ()

  Returns True; this method will be reimplemented in container types where it may return False.
- bool intp ()

  Returns False; this method is reimplemented in other types and will return True if the given expression can be converted to an integer.

- AbstractIterator iterator ()

  Returns an iterator object for the value; the default iterator object returned is SingleValueIterator.
- int lsize ()

  Returns 1; the return value of this method should give the list size of the value, which is normally 1 for non-lists (except for NOTHING where the size will be 0) and the number of the elements in the list for lists; this method will be reimplemented in other types where it may return other values.
- int size ()

  Returns zero; this method will be reimplemented in container types where it may return a non-zero value.
- bool sizep ()

  Returns True if the type can return a non-zero size (True for containers including binary objects and strings, False for everything else)
- bool strp ()

  Returns False; this method is reimplemented in other types and will return True if the given expression can be converted to a string.
- bool toBool ()

  Returns the boolean representation of the value; the default is False.
- float toFloat ()

  Returns the floating-point representation of the value; the default is 0.0.
- int toInt ()

  Returns the integer representation of the value; the default is 0.
- number toNumber ()

  Returns the arbitrary-precision numeric representation of the value; the default is 0.
- string toString ()

  Returns the string representation of the value; the default is an empty string.
- string type ()

  Returns the string type for the value.
- int typeCode ()

  Returns the type code for the value.
- bool val ()

  Returns False; this method is reimplemented in other types and will return True if the given expression has a value that would be converted to True according to the rules described in %perl-bool-eval.

45.68.1 Detailed Description

Methods in this pseudo-class are available to be executed on any value type (even NOTHING); this is the root class for all pseudo-classes.

45.68.2 Member Function Documentation

45.68.2.1 bool Qore::zzz8valuezzz9::callp ( )

Returns False; this method is reimplemented in other types and will return True if the given expression is a callable value (ie closures or call references)

Returns

False; this method is reimplemented in other types and will return True if the given expression is a callable value (ie closures or call references)

Code Flags:

CONSTANT

Example:

```
if ($n.callp())
    printf("the reuslt of calling the value: %y\n", $n());
```

45.68.2.2 bool Qore::zzz8valuezzz9::empty ( )

Returns True; this method will be reimplemented in container types where it may return False.

This pseudo-method will return False in all non-container types; use Qore::zzz8valuezzz9::val() to check if a generic expression that might not be a container type has a value instead.

Returns

True; this method will be reimplemented in container types where it may return False

Code Flags:

CONSTANT

Example:

```
my int $i = 100;
if ($i.empty())
    printf("%y: is empty (probably because it's a value of type %y)\n", $i, $i.type());
```

See also

Qore::zzz8valuezzz9::val()

45.68.2.3 bool Qore::zzz8valuezzz9::intp ( )

Returns False; this method is reimplemented in other types and will return True if the given expression can be converted to an integer.

Returns

False; this method is reimplemented in other types and will return True if the given expression can be converted to an integer

Code Flags:

CONSTANT

Example:

```
if ($n.intp())
    printf("%y: can be converted to an integer: %d\n", $n, int($n));
```

See also

Qore::zzz8stringzzz9::intp() for an example of an implementation of this method where the result depends on the value and not the type

45.68.2.4 AbstractIterator Qore::zzz8valuezzz9::iterator ( )

Returns an iterator object for the value; the default iterator object returned is SingleValueIterator.

Returns

an iterator object for the value; the default iterator object returned is SingleValueIterator

Code Flags:

CONSTANT

Example:

```
map printf("+ %y\n", $1), $v.iterator();
```

Since

Qore 0.8.6

45.68.2.5 int Qore::zzz8valuezzz9::lsize ( )

Returns 1; the return value of this method should give the list size of the value, which is normally 1 for non-lists (except for NOTHING where the size will be 0) and the number of the elements in the list for lists; this method will be reimplemented in other types where it may return other values.

Returns

1; the return value of this method should give the list size of the value, which is normally 1 for non-lists (except for NOTHING where the size will be 0) and the number of the elements in the list for lists; this method will be reimplemented in other types where it may return other values

Code Flags:

CONSTANT

Example:

```
printf("iterating %d element%s\n", $val.lsize(), $val.lsize() == 1 ? "" : "s");
foreach my any $element in ($val) {
    do_something($element);
}
```

45.68.2.6 int Qore::zzz8valuezzz9::size ( )

Returns zero; this method will be reimplemented in container types where it may return a non-zero value.

This pseudo-method will return 0 in all non-container types; use Qore::zzz8valuezzz9::val() to check if a generic expression that might not be a container type has a value instead.

Returns

zero; this method will be reimplemented in container types where it may return a non-zero value

Code Flags:

CONSTANT

Example:

```
my int $i = 100;
if (!$i.size())
    printf("%y: has size 0 (probably because it's a value of type %y)\n", $i, $i.type());
```

See also

- Qore::zzz8valuezzz9::size()
- Qore::zzz8valuezzz9::val()

45.68.2.7 bool Qore::zzz8valuezzz9::sizep ( )

Returns True if the type can return a non-zero size (True for containers including binary objects and strings, False for everything else)

Code Flags:

CONSTANT

Returns

True if the type can return a non-zero size (True for containers including binary objects and strings, False for everything else)

See also

Qore::zzz8valuezzz9::size()

Since

Qore 0.8.9

45.68.2.8 bool Qore::zzz8valuezzz9::strp ( )

Returns False; this method is reimplemented in other types and will return True if the given expression can be converted to a string.

Returns

False; this method is reimplemented in other types and will return True if the given expression can be converted to a string

Code Flags:

CONSTANT

Example:

```
if ($n.strp())
    printf("%y: can be converted to a string: '%s'\n", $n, string($n));
```

45.68.2.9 bool Qore::zzz8valuezzz9::toBool ( )

Returns the boolean representation of the value; the default is False.

Code Flags:

CONSTANT

Returns

the boolean representation of the value; the default is False

Note

if Qore::zzz8valuezzz9::intp() returns True, then the value can also be converted to a boolean value

Since

Qore 0.8.6

45.68.2.10 float Qore::zzz8valuezzz9::toFloat ( )

Returns the floating-point representation of the value; the default is 0.0.

Code Flags:

CONSTANT

Returns

the floating-point representation of the value; the default is 0.0

Note

if Qore::zzz8valuezzz9::intp() returns True, then the value can also be converted to a floating-point value

Since

Qore 0.8.6

45.68.2.11 int Qore::zzz8valuezzz9::toInt ( )

Returns the integer representation of the value; the default is 0.

Code Flags:

CONSTANT

Use Qore::zzz8valuezzz9::intp() to determine if the current value can be converted to an integer

Returns

the integer representation of the value; the default is 0

Since

Qore 0.8.6

45.68.2.12 number Qore::zzz8valuezzz9::toNumber ( )

Returns the arbitrary-precision numeric representation of the value; the default is 0.

Code Flags:

CONSTANT

Returns

the integer representation of the value; the default is 0

Note

if Qore::zzz8valuezzz9::intp() returns True, then the value can also be converted to an arbitrary-precision numeric value

Since

Qore 0.8.8

45.68.2.13 string Qore::zzz8valuezzz9::toString ( )

Returns the string representation of the value; the default is an empty string.

Code Flags:

CONSTANT

Use Qore::zzz8valuezzz9::strp() to determine if the current value can be converted to a string

Returns

the string representation of the value; the default is an empty string

Note

check pseudo-classes for each type for other variants of this method; some may take arguments to affect the output format

Since

Qore 0.8.6

45.68.2.14 string Qore::zzz8valuezzz9::type ( )

Returns the string type for the value.

Returns

the string type for the value; see String Type Constants for possible return values for each type

Code Flags:

CONSTANT

Example:

```
my string $t = $n.type();
```

Note

It is faster and more efficient to use Qore::zzz8valuezzz9::typeCode() for comparing data types

45.68.2.15 int Qore::zzz8valuezzz9::typeCode ( )

Returns the type code for the value.

This method is recommended over Qore::zzz8valuezzz9::type() or the type(any) or typename(any) functions for comparing data types as it is much faster and more efficient than the other alternatives (which work with string values instead of integer codes).

Returns

the type code for the value; see Type Code Constants for possible return values for each type

Code Flags:

CONSTANT

Example:

```
switch ($v.typeCode()) {
    case NT_INT:
        printf("%y: is an integer\n", $v);
        break;

    case NT_DATE:
        printf("%y: is a date\n", $v);
        break;

    default:
        printf("%y: is something else entirely\n", $v);
        break;
}
```

45.68.2.16 bool Qore::zzz8valuezzz9::val ( )

Returns False; this method is reimplemented in other types and will return True if the given expression has a value that would be converted to True according to the rules described in %perl-bool-eval.

This pseudo-method can be used to mimic perl's boolean evaluation when %strict-bool-eval is enabled; the return value of this method is the same as that of any value when evaluated in a boolean context when %perl-bool-eval is enabled (the default).

Prior to version 0.8.6, Qore implemented strict mathematical boolean evaluation by default (where any value must evaluate to a non-zero integer or floating-point value to be True), so this pseudo-method was implemented to provide the same functionality in qore. Note that as of qore 0.8.6, %perl-bool-eval is the default.

Returns

False; this method is reimplemented in other types and will return True if the given expression has a value; the return value of this method is the same as that of any value when evaluated in a boolean context when %perl-bool-eval is enabled (the default)

Code Flags:

CONSTANT

Example:

```
if ($n.val())
    printf("%y: has a value\n", $n);
```

Note

use Exists Operator (exists) to tell if an expression has a value or not

See also

- %perl-bool-eval
- %strict-bool-eval

Index