RC Lib

Version 201607061602

Generated by Doxygen 1.8.11

Contents

Chapter 1

RC Lib

1.1 Download

You can download the latest version of RC Lib as a zip or tar.xz. (Or you can obtain older versions.)

The latest version of this documentation can be found online.

1.2 Introduction

RC Lib is a cross-platform C++ programming library designed to significantly improve the way you design and structure programs. It seems to be generally accepted that one must choose a compromise between runtime performance and syntactical convenience for the programmer. This library is designed with the philosophy that this compromise is not necessary. After programming for quite some time, I found that my programming tasks largely settled into two categories. If I needed something high performance or with a complicated architecture, I would write it in C++. And if I wanted to quickly prototype an idea, or quickly parse some files or text, I would write in a higher level language like perl. When a problem fell in the uncomfortable middle ground, neither approach felt quite right.

RC Lib seeks to provide a seamless integration of these approaches by providing high level language features for parsing text, working with files, and structuring data, while preserving the structural advantages and characteristic performance of C++. Whenever possible syntactic sugar for usage of the library was favored over the simplicity of the implementation.

A core design goal of RC Lib is the ease of writing robustly correct and secure programs. Where a bound can be checked, it is checked. Where a pointer can be verified as non-NULL, it is verified. The idea that libraries should avoid validating inputs in the name of efficiency, as is done throughout the STL, is an antiquated notion which has resulted in a great deal of harm. It is the responsibility of the compiler to optimize away input validation when it can be determined to not be necessary, and in fact modern compilers often do an excellent job of this. With branch prediction in modern cpu's, the impact of remaining validation is negligible. With this in mind, RC Lib is designed from the ground up to favor ease of writing correct code.

RC Lib primarily targets cross-platform Linux and Windows development, although presumably it would also compile on OS X with minimal problems. (If anyone wishes to be responsible for OS X testing, please contact me.)

Lastly, RC Lib is provided freely and with the permissive Boost license, to remove barriers to its use.

1.3 File handling

There is a fundamental philosophical difference between the STL string and stream classes and the RC Lib string and file classes. The STL performs character formatting and parsing at the input and output classes, while the string class handles only basic storage and character manipulation. While the STL streams provide an overload interface with "<<" and ">>" for serialized character data, for binary data the stream read/write interface works only with a byte array. In RC Lib, the responsibility for formatting of character data and conversion of other types to character data is centralized in the string class (RC::RStr), while the file classes (RC::FileRead, RC::FileWrite, and RC::FileRW) use overloading of Get and Put to process binary data of various plain old data types as well as complex classes.

1.4 String manipulation

The string manipulation features of RC Lib are centralized in RC::RStr, which wraps and augments std::string. RStr has constructors for converting all of the primitives to strings, along with all of the formatting options for base and precision. It also has a series of Get calls for parsing out primitives. For aligning and formatting text there are a series of Pad and Wrap functions. RStr also has syntactically elegant regular expression matching and substitution. (These wrap the C++11 implementations, or for pre-C++11 can make use of the Boost regular expression libraries.)

RStr also provides high level capabilities like splitting and joining strings. For example SplitWords will return a one dimensional structure of all the words. Also some more specialized routines are available for working with comma-separated values, base64 encoding, and long hex strings.

An additional advantage of keeping string manipulation in the string class rather than in streams arises with multi-threaded programming, in which concurrent stream output calls can be interwoven. Thus it is prudent to consistently prepare the entire formatted string within the originating thread, and then output it atomically.

1.5 Error handling

A common infrastructure for error handling is used throughout RC Lib of throwing RC::ErrorMsg exceptions, or sub-types like RC::ErrorMsgFile. The ErrorMsg class provides a cross-platform way to achieve a stack trace for any unexpected errors. If an ErrorMsg is allowed to propagate through to program termination, this will result in an informative stack trace of the error being printed to stderr, facilitating debugging. Alternatively, programs can capture and display or log this stack trace.

With traditional development protocols there is often a huge disconnect between users in the field experiencing bugs, and developers in a debugger attempting to reproduce them. Bugs that require specific conditions can often go years without being fixed. The RC::ErrorMsg approach attempts to resolve that by assisting users in providing debugging-useful error messages when unhandled exceptional events occur.

1.6 Smart pointers

There are three approaches to smart pointers in RC Lib. The basic pointer class, RC::Ptr, operates as a straight templated substitute for the classic C pointer, but ensures the pointer defaults to NULL, and provides a debuggable ErrorMsgNull exception if a NULL dereference is accidentally attempted. This should be used liberally wherever a C pointer would otherwise be used, since RC::Ptr is easily optimized away when safe to do so, and it casts both implicitly and explicitly to a C pointer.

Automatic memory management is provided by RC::APtr, which is a reference counting pointer that deletes the object automatically when the last copy of APtr leaves scope. The reference counted pointer is implicitly shared

every time an APtr is assigned or copy-constructed from another APtr (in a thread-safe manner with C++11). Any place you find a manual delete call in your code, try replacing it with a properly scoped APtr to make all of your memory management automatic.

Self-referential pointers can be managed with RC::RevPtr, which is a shared pointer that can be made to "revoke" itself, or make all shared copies NULL, when the object containing the master copy leaves scope. In this way a pointer can be safely obtained to an object which may be deleted, such that the pointer will become NULL when the object no longer exists. Use this to immunize against dangling pointers.

1.7 Data structures

For basic array storage, RC::Data1D, RC::Data2D, and RC::Data3D provide features comparable to std::vector, but with built-in bounds checking, more efficient memory management in numerous areas, and the ability to access and extract a raw C array corresponding to the managed storage. The Data#D containers each contain lightweight resizing and offset operations, meaning they can be made to temporarily refer only to portions of the data. When an array must be generated for working with a C library, the Raw and Extract methods permit interfacing these rather than falling back to problem-prone C-style memory management. Conversely Data1D can be made to wrap an externally obtained C-style array, providing bounds checking and the other features. In tight loops, the bounds checking of these structures is often optimized away.

Similar features are provided by the RC::Bitfield, RC::Bitfield2D, and RC::Bitfield3D classes, but for efficiently packed arrays of bits. In contrast to std::vector's specialization of bool, Data1D<bool> has underlying storage of precisely what it says, an array of bools.

The RC::Tuple class provides features analogous to std::tuple, but with the added ability to Apply the Tuple contents as parameters to a function call, cast them all to a fixed type Data1D, or implicitly construct them during a function return. For example, Tuple<int, RStr> f() { return {3, "foo"}; }

1.8 Syntactic sugar

Short-hand consistent types are provided in Types.h of i8, u8, i16, u16, i32, u32, i64, u64, f32, f64, and when available, f80 and f128. Those beginning with i are signed integers, with u unsigned integers, and with f floating point types, while the number following is of course the bit size. This provides a more predictable cross-platform sizing than the default types in a uniform and syntactically pleasing shorthand.

Conditional comparisons are made more elegant by the helper methods RC::Betw and RC::OneOf. Rather than "if (3.14 < x && x <= 9)" one can write the more natural "if (3.14 < Betw(x) <= 9)", which remains valid for any type combinations with comparators defined. Comparisons with lists are made more natural by the variadic template method OneOf, which has a syntax like "if (str == OneOf("foo", "bar", "baz", str2))". OneOf remains valid for any combination of types for which the variables passed to OneOf can be constructed as the type of the variable they're comparing to.

RC Lib also provides an update of the C-style main function into a C++ one, with the RC_MAIN macro, used like "RC_MAIN { /* body of main here */; return 0 }". In it, the traditional "int argc, char *argv[]" variables are replaced by an "RC::Data1D<RC::RStr> args" which is bounds checked and coupled to all of the string parsing features of RStr. Additionally RC_MAIN registers a segfault handler which converts segfaults into a stack tracing RC::ErrorMsgFatal exception.

1.9 Miscellaneous convenience

Efficient Mersenne Twister random number generation is provided by RC::RND, which includes Get∗ functions for converting the randomness to the most common types of random data. True random number generation is provided more slowly by RC::EntropyRND which generates randomness on demand by monitoring the jitter of the high precision clock.

Cross-platform high precision timing and sleeping is provided by RC::Time. This class also provides a simplified and consistent method for working with the system date and time, as well as formatting and parsing date-time strings.

Debugging can be done more efficiently if rather than stepping through code inside of a debugger, one inserts the RC_DEBOUT() macro at critical locations. This macro takes no parameters or a list of variables, and outputs to stderr the filename, the line number, the name of each variable, and what its value is. It then flushes stderr to ensure output is complete before any subsequent program termination. This allows a low effort approach to rapidly tracing the flow of the program state, rather than slowly stepping through the program at a halted state.

1.10 Metaprogramming

Metaprogramming can provide significant increases in programmer efficiency and code maintainability by reducing code repeats. (But it must be used judiciously.)

An assortment of metaprogramming macros are provided in Macros.h, all defined with the macro-namespace convention of prefixing with RC_. For example, RC_ARGS_EACH(Macro,a,b,c,...) –> Macro(a)Macro(b)Macro(c)... provides the ability to wrap a list of parameters in a macro you define, which can be used to generate repeated sections of code on lists of types or objects. Similar capability is provided by RC_ARGS_LIST but for modifing a list of function parameters in place. RC_ARGS_PAIR provides the same capability but for type/value pairs. A number of other macro primitives are provided to facilitate writing your own metaprogramming macros.

A safer alternative to std::function (with stack-tracing exceptions) is provided with RC::Caller, which provides a wrapper for functors, static methods, and member methods. The member method constructor can receive an assigned object so that a single well-defined type Caller can be passed for a call to either a static method or a member method on a pre-assigned object. This facilitates passing callback handlers to a specific object without the signalling routine needing knowledge of the receiving object's type. For type-agnostic function handling, any templated Caller can be wrapped by an untemplated DynCaller, allowing arrays of methods of different types, and safe runtime extraction.

1.11 Project dependency

One of the most common reasons given to avoid using any particular library in a project is the concern over introducing a new dependency to the source code and/or the executable. This is often a legitimate concern, as external changes to a library can often break a piece of code developed in-house, and the use of any shared library requires it to be kept synchronized with the version used in development. RC Lib intentionally avoids this problem by being a fully header-only library. When you use RC Lib for a short prototype program, perhaps you use a shared copy of the include files. But when you use RC Lib in a project, you do not introduce a project dependency. You simply copy the RC directory into your source tree as-is, and it becomes an integral and stable part of your program. The permissive license provided was chosen to make this easy to do.

1.12 License

Continue to the Examples page.

Chapter 2

Examples

```
// tolower.cpp - Take one string from the command line and make it lowercase.

#include <RC/RC.h>

// This provides using namespace for RC and std.
// It is recommended to use it for fast prototyping, but only in cpp files.
RC_USE

// This wraps argc and argv as RC::Data1D<RC::RStr> args.
RC_MAIN {
  if (args.size() < 2) {
    cerr << args[0] << " <string>\n";
    return -1;
  }

  RC::RStr str = args[1];

  // For each character
  for (char &c : str) {
    // If it's between 'A' and 'Z' inclusive
    if ('A' <= Betw(c) <= 'Z') {
      // Make it lowercase.
      c += 'a' - 'A';
    }
  }

  cout << str << endl;

  return 0;
}

```

```
// battery.cpp - Determine the battery state from the Linux power_supply files.

#include "RC/RC.h"

RC_USE

const RStr dir_base = "/sys/class/power_supply/";

// Read a line from a file.
RStr GetFile(const RStr& filename) {
  // Concatenate the strings.
  RStr pathname = dir_base + filename;

  // Open the file for reading.
  FileRead br(pathname);

  // Get one line as an RStr
  RStr retval;
  br.Get(retval);

  return retval;
}

RC_MAIN {
  // Output whether or not the AC is plugged in.
  // The RC::RStr::Get_bool checks for true, T, or a non-zero number.
  cout << "AC is " << (GetFile("AC/online").Get_bool() ? "on" : "off") << endl;
```

```
28
29   // Read the current, full, and design-peak energy levels.
30   // Unsigned integers are extracted from the RStr.
31   u64 current = GetFile("BAT0/energy_now").Get_u64();
32   u64 lastfull = GetFile("BAT0/energy_full").Get_u64();
33   u64 designfull = GetFile("BAT0/energy_full_design").Get_u64();
34
35   // Calculate the percent full the battery is.
36   u32 perc = 100 * current / (f64) lastfull + 0.5;
37   // Calculate how close to ideal the battery is.
38   f64 quality = lastfull / (f64) designfull;
39
40   cout << "Battery:  " << perc << "%" << endl;
41   cout << "Quality:  " << RStr(quality,FIXED,2) << endl;
42
43   return 0;
44 }
```

```
1 // randomize_lines.cpp - Shuffle the lines from stdin.
2
3 #include <RC/RC.h>
4
5 RC_USE
6
7 RC_MAIN {
8   size_t max_lines_out = -1;  // Max size_t
9
10   // If there are more arguments than the program name.
11   if (args.size() > 1) {
12     // Check to see if a number was given.
13     if (args[1].Is_u64()) {
14       // If so, get it.
15       max_lines_out = args[1].Get_u64();
16     }
17     else {
18       // If the argument was not a number, print the help.
19       cerr << args[0] << " [max_lines_out]\n";
20       return -1;
21     }
22   }
23
24   // Open stdin as a FileRead
25   FileRead fr(stdin, false);
26
27   // Get all the lines until end-of-file.
28   Data1D<RStr> lines;
29   fr.GetAll(lines);
30
31   // Randomize the order of the lines.
32   lines.Shuffle();
33
34   // Shrink the array if we should output fewer lines.
35   // Note this does not reallocate.
36   if (lines.size() > max_lines_out) {
37     lines.Resize(max_lines_out);
38   }
39
40   // Output the remaining lines.
41   for (RStr line : lines) {
42     cout << line << endl;
43   }
44
45   return 0;
46 }
47
```

```
1 // parse_and_sum.cpp - Parse and process some comma-separated values.
2
3 #include <RC/RC.h>
4
5 RC_USE
6
7 RC_MAIN {
8   RStr numstr = "3.14, 5.7, 8, 12";
9
10   // Split the comma-separated values into an array.
11   Data1D<RStr> str_arr = numstr.SplitCSV();
12
13   // Sum the array of numbers
14   f64 sum = 0;
15   for (RStr x : str_arr) {
16     sum += x.Get_f64();  // Convert to a 64-bit float.
17   }
18
```

```
  // Outputs "3.14 + 5.7 + 8 + 12 = 28.84"
  cout << RStr::Join(str_arr, " + ") << " = " << sum << endl;

  return 0;
}
```

```
// random_average.cpp - Generate and average 5000 numbers.

#include <RC/RC.h>

RC_USE

RC_MAIN {
  // Seeds the random number generator with the high precision clock.
  RND rng;
  // Initializes the timer.
  Time timer;

  // Appends 5000 random 64-bit floats from [0,1) to vals.
  Data1D<f64> vals;
  for (u64 r=0; r<5000; r++) {
    vals += rng.Get_f64();;
  }

  // Calculates the average.
  f64 avg=0;
  for (size_t i=0; i<vals.size(); i++) {
    avg += vals[i];
  }
  avg /= vals.size();

  // Reports how long it took, to a fraction of a second.
  cout << "It took " << timer.SinceStart() << " seconds.\n";

  // A flushed stderr printout of "random_average.cpp:30, avg = 0.496605"
  RC_DEBOUT(avg);

  // Reports if the avg was in an acceptable range.
  cout << "The average was " << (0.48 < Betw(avg) < 0.52 ? "good\n" : "bad\n");

  return 0;
}
```

```
// callback.cpp - Processes a callback with a Caller, and returns a Tuple.

#include <RC/RC.h>

RC_USE

class Doer {
  public:

  Tuple<u16, RStr> DoStuff(Caller<void, f32> callback) {
    RND rng;
    f32 val = rng.Get_f32();

    // Because of what was passed in, this will call w.CallBack, even
    // though it knows nothing of the Waiter class.
    callback(val);

    // Implicitly returns a Tuple.
    return {rng.Get_u16(), "apples"};
  }
};


class Waiter {
  public:

  void CallIt(Doer &d) {
    // Pass a Caller<void, f32> to DoStuff, and receive a Tuple back.
    auto tup = d.DoStuff(MakeCaller(this, &Waiter::CallBack));

    // Alternatively one can construct the Caller like this.
    auto tup2 = d.DoStuff({this, &Waiter::CallBack});

    // Prints a value like "It returned 51813 apples"
    cout << "It returned " << tup.Get<0>() << " " << tup.Get<1>() << endl;
  }

  // Receives a call back from DoStuff
  void CallBack(f32 x) {
    cout << "I got x == " << x << endl;
  }
```

```
42 };
43
44
45 RC_MAIN {
46   Doer d;
47   Waiter w;
48
49   w.CallIt(d);
50
51   return 0;
52 }
53
```

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 RC Namespace Reference

Classes

- class APtr

  A reference counting ptr that auto-deletes what it points to when the last copy leaves scope or is otherwise destructed.
- class BaseRND

  An abstract class which provides functions for obtaining randomness in convenient forms.
- class Bitfield

  A bounds-safe one-dimensional vector-like structure of efficiently packed bits.
- class Bitfield2D

  A bounds-safe two-dimensional structure of efficiently packed bits.
- class Bitfield3D

  A bounds-safe three-dimensional structure of efficiently packed bits.
- class Caller

  A general purpose function class which can refer to any static method, member method, functor, or lambda function.
- class Data1D

  A bounds-safe one-dimensional vector-like structure.
- class Data2D

  A bounds-safe two-dimensional resizeable structure.
- class Data3D

  A bounds-safe three-dimensional resizeable structure.
- class DebugTrack

  Inherit this class to add construction, destruction, and assignment output tracking.
- class DynCaller

  A typeless container for a Caller, which has methods for dynamically casting it to the correct type.
- class Endian

  Auto-detects the endianness of the compilation target, and provides automatic endian conversion features.
- class EntropyRND

  Provides true random numbers sourced from environmental noise.
- class ErrorMsg

  An exception class that records where the exception was thrown and provides a stack trace.
- class ErrorMsgBounds

  A subtype of RC::ErrorMsg for Bounds errors.
- class ErrorMsgCast

A subtype of RC::ErrorMsg for Bad Cast errors.

- class ErrorMsgFatal

  A subtype of RC::ErrorMsg for Fatal errors.
- class ErrorMsgFile

  A subtype of RC::ErrorMsg for File related errors.
- class ErrorMsgMemory

  A subtype of RC::ErrorMsg for Memory errors.
- class ErrorMsgNet

  A subtype of RC::ErrorMsg for Networking related errors.
- class ErrorMsgNull

  A subtype of RC::ErrorMsg for Null errors.
- class File

  A class with static methods for file and directory info and manipulation.
- class FileBase

  Provides the common methods for the FileRead/FileWrite/FileRW classes.
- class FileRead

  A file reading class that provides buffered and unbuffered access to files with support for non-POD classes.
- class FileRW

  A file class for both reading and writing that provides buffered and unbuffered output to files.
- class FileWrite

  A file writing class that provides buffered and unbuffered output to files with support for non-POD classes.
- class HoldRelated

  Stores the value of type Hold while providing access via type Provide.
- class LoopIndex

  A size_t like integer class which automatically stays within its range.
- class Net

  Provides both client and server sides of blocking TCP connections.
- class PluralStr

  Provides number-based singular/plural string management.
- class Ptr

  A safe pointer class that throws an RC::ErrorMsgNull if a null dereference is attempted.
- class RAIter

  A bounds-checked iterator for random-access containers that knows when its target was deleted.
- class RevPtr

  A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.
- class RND

  A Mersenne Twister random number generator class with integer, array, or time-based seeding.
- class RStr

  A bounds-safe string class which provides an identical interface to std::string plus many convenience functions for string manipulation.
- class Segfault

  A static class for catching and throwing segfaults.
- class Sock

  A portable socket interface for reading and writing to an open socket.
- class Time

  Accesses and formats the system date and time, and provides high precision timing.
- class TimeOfDay

  A class which obeys periodic time-of-day boundaries, and can manage times being within a daily time frame.
- class Tuple

  An efficient Tuple class with Set, Get, and an Apply function to pass the tuple contents on to any function.
- class UntypedCaller

  The base class of Caller without return type or parameters specified.
- class URand

  Cryptographically strong RNG, uses /dev/urandom.

Typedefs

- typedef RAIter< RStr, char > RStrIter

  The random-access iterator for RStr begin() and end()
- template<class FT >
  using FuncPtr = FT *

  Clean C-style function pointers: FuncPtr<void(int)> f = &func; f(4);.
- template<class C , class FT >
  using MemFuncPtr = FT(C::*)

  Clean function pointer syntax for member function.

Enumerations

- enum WriteMode { **TRUNCATE** =0, **KEEP**, **APPEND**, **NEWONLY** }

  Valid write modes for a FileWrite or FileRW.
- enum RStr_IntStyle {
  DEC =100000, **HEX**, **HEX0x**, **OCT**,
  OCT0, **BIN**, **CHAR** }

  The styles in which an integral number can be formatted.
- enum RStr_FloatStyle { **AUTO** =100000, **FIXED**, **SCI** }

  The styles in which a floating point number can be formatted.

Functions

- template<class T >
  std::ostream & operator<< (std::ostream &out, APtr< T > obj)

  A convenience stream output for displaying the enclosed \ object.
- std::ostream & operator<< (std::ostream &out, const Bitfield &b)

  Outputs data to a stream as a character series of '1's and '0's.
- std::ostream & operator<< (std::ostream &out, const Bitfield2D &b)

  Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2.
- std::ostream & operator<< (std::ostream &out, const Bitfield3D &b)

  Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2, and double-spaces after entries of dimension 3.
- template<class C , class Ret , class... Params>
  Caller< Ret, Params... > MakeCaller (C *obj, Ret(C::*func)(Params...))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class Ret , class... Params>
  Caller< Ret, Params... > MakeCaller (C &obj, Ret(C::*func)(Params...))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class Ret , class... Params>
  Caller< Ret, C &, Params... > MakeCaller (Ret(C::*func)(Params...))

  Generates a Caller for the member function with type inference, inserting a first parameter with a reference to the object.
- template<class Ret , class... Params>
  Caller< Ret, Params... > MakeCaller (Ret(*func)(Params...))

  Generates a Caller for the specified static function, with type inference.
- template<class Ret , class... Params, class Functor >
  Caller< Ret, Params... > MakeFunctor (Functor func)

  A special generator for functors, which requires specifying the return type and arguments.

- template<class C , class MemberFunc >
  auto MakeCaller (Ptr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class MemberFunc >
  auto MakeCaller (APtr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class MemberFunc >
  auto MakeCaller (RevPtr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.
- std::ostream & operator<< (std::ostream &out, const Data1D< char > &d)

  Outputs data to a stream as { a, b, ... }.
- std::ostream & operator<< (std::ostream &out, const Data1D< u8 > &d)

  Outputs data to a stream as { 61, 62, ... }.
- std::ostream & operator<< (std::ostream &out, const Data1D< i8 > &d)

  Outputs data to a stream as { 61, -42, ... }.
- template<class T >
  std::ostream & operator<< (std::ostream &out, const Data1D< T > &d)

  Outputs data to a stream as { elem0, elem1, ... }.
- template<class T >
  void swap (RC::Data1D< T > &a, RC::Data1D< T > &b)

  Efficiently swap all the contents of a and b.
- template<class T , size_t N>
  auto MakeData1D (T(&arr)[N]) -> RC::Data1D< T >

  Convenience generator for safely converting C-style arrays.
- template<class T >
  std::ostream & operator<< (std::ostream &out, const Data2D< T > &d)

  Outputs data to a stream as { { x_0_0, x_1_0, ...}, { x_0_1, x_1_1, ...}, ... }.
- template<class T2 >
  void swap (Data2D< T2 > &a, Data2D< T2 > &b)

  Efficiently swap all the contents of a and b.
- template<class T >
  std::ostream & operator<< (std::ostream &out, const Data3D< T > &d)

  Outputs data to a stream.
- template<class T2 >
  void swap (Data3D< T2 > &a, Data3D< T2 > &b)

  Efficiently swap all the contents of a and b.
- template<class T >
  bool FileGetWrapper (FileRead &fr, T &data)

  Overload this with a specialization to support Get on a custom type.
- template<class T >
  void FilePutWrapper (FileWrite &fw, const T &data)

  Overload this with a specialization to support Put on a custom type.
- template<>
  bool FileGetWrapper< std::string > (FileRead &fr, std::string &data)

  A specialization for handling std::string objects.
- template<>
  void FilePutWrapper< std::string > (FileWrite &fw, const std::string &data)

  A specialization for handling std::string objects.
- template<class Cont , class T >
  RAIter< Cont, T > operator+ (size_t x, const RAIter< Cont, T > &iter)

  Returns a new iterator with an offset incremented by x.
- template<class T >
  void UnusedVar (const T &)

Mark an unused variable to suppress warnings.

- template<class T >
  std::ostream & operator<< (std::ostream &out, Ptr< T > obj)

  A convenience stream output for displaying the enclosed \ object.
- template<class T >
  BetwCompare< T > Betw (const T &x)

  Returns a comparator that can be used for range comparisons.
- template<class... Args>
  OneOfCompare< Args... > OneOf (Args...args)

  Returns a comparator that can be used to check if something is in a list.
- template<class T >
  RangeHelper< T > Range (const T &start, const T &past_the_end)

  Provides an iterator which dereferences to the iterated values [start,past_the_end).
- template<class T2 >
  auto IndexOf (const T2 &cont) -> RangeHelper< decltype(cont.size())>

  Provides an iterator which dereferences to the indices of cont from [0,cont.size()).
- template<class T >
  std::ostream & operator<< (std::ostream &out, RevPtr< T > obj)

  A convenience stream output for displaying the enclosed \ object.
- u64 RND_Get_Range (u64 range)

  Uses RND as a RandomNumberGenerator, e.g. for random_shuffle.
- u64 EntropyRND_Get_Range (u64 range)

  Uses EntropyRND as a RandomNumberGenerator, e.g. for random_shuffle.
- u64 URand_Get_Range (u64 range)

  Uses URand as a RandomNumberGenerator, e.g. for random_shuffle.
- RND & RND_Gen ()

  This returns a singleton of RND .
- EntropyRND & EntropyRND_Gen ()

  This returns a singleton of EntropyRND .
- URand & URand_Gen ()

  This returns a singleton of URand .
- RStr operator+ (const RStr &lhs, const RStr &rhs)

  Concatenates two RStr'ings.
- std::istream & operator>> (std::istream &in, RStr &rstr)

  Input text from a std::istream (e.g., std::cin) into an RStr.
- std::ostream & operator<< (std::ostream &out, const RStr &rstr)

  Output text from an RStr to a std::ostream (e.g., std::cout).
- void swap (RStr &lhs, RStr &rhs)

  Swaps two RStr'ings.
- std::istream & getline (std::istream &is, RStr &str, char delim='\n')

  Sets str equal to one line from the input stream, up to end of file or the delim, which is discarded.
- bool operator== (const RStr &lhs, const RStr &rhs)

  True if lhs equals rhs.
- bool operator!= (const RStr &lhs, const RStr &rhs)

  True if lhs does not equal rhs.
- bool operator< (const RStr &lhs, const RStr &rhs)

  True if lhs is less than rhs.
- bool operator> (const RStr &lhs, const RStr &rhs)

  True if lhs is greater than rhs.
- bool operator<= (const RStr &lhs, const RStr &rhs)

  True if lhs is less than or equal to rhs.

- bool operator>= (const RStr &lhs, const RStr &rhs)

  True if lhs is greater than or equal to rhs.
- const RStr REG_FLT ("[-+]?[0-9]*\\.?[0-9]+[eE][-+]?[0-9]+|[-+]?[0-9]*\\.?[0-9]+")

  A regular expression component to match a floating point number.
- const RStr REG_FLTP ("([-+]?[0-9]*\\.?[0-9]+[eE][-+]?[0-9]+|[-+]?[0-9]*\\.?[0-9]+)")

  A regular expression component to match and return a floating point number.
- template<class... Types>
  Tuple< Types... > MakeTuple (Types...types)

  A convenience generator to make a Tuple from the given elements with type inference.
- template<class T >
  bool IsSignedType (const T &x __attribute__((unused))=0)

  True if the type is signed.
- template<class T >
  bool IsIntegerType (const T &x __attribute__((unused))=1)

  True if the type is an integer type.
- template<class T >
  void AssertFloat ()

  For generating compilation errors if the type is not a float.
- template<class T >
  void AssertInteger ()

  For generating compilation errors if the type is not an integer.
- template<class T >
  auto ForceFloat (T val) -> typename ForceFloatHelper< T, FloatType< T >::value >::type

  If T is a float type, return the same type. Otherwise return it as an f64.
- template<class T >
  std::make_unsigned< T >::type ForceUnsigned (T val)

  Returns the unsigned equivalent to the given type.
- template<class T >
  std::make_signed< T >::type ForceSigned (T val)

  Returns the signed equivalent to the given type.
- template<class T >
  T MIN_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::min())

  Provide the minimum value held by this type. For floats, the smallest positive value.
- template<class T >
  T MAX_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::max())

  Provide the maximum value held by this type.
- template<class T >
  T LOW_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::lowest())

  Provide the lowest value held by this type, for which no value is less.
- template<class T >
  T MIN_POS (const T &x __attribute__((unused))=MIN_POS_Helper< T, FloatType< T >::value >::F())

  Provide the minimum positive value held by this type.

Variables

- const size_t ErrorMsg_text_bufsize = 4096

  The maximum size of the char array returned by ErrorMsg::what() and ErrorMsg::GetError(), including the null.
- const size_t ErrorMsg_type_bufsize = 256

  The maximum size of the char array returned by ErrorMsg::GetType(), including the null.

7.1.1 Detailed Description

Note: RC_EMPTY, RC_ARGS_NUM, RC_ARGS_EACH, RC_ARGS_BET, RC_ARGS_LIST, and RC_DEBOUT require each of their parameters to begin with a letter, number, or underscore.

7.1.2 Typedef Documentation

7.1.2.1 template<class C , class FT > using RC::MemFuncPtr = typedef FT(C::*)

Clean function pointer syntax for member function.

MemFuncPtr<A, void(int)> f = &A::func; A a; (a.*f)(4);

7.1.3 Enumeration Type Documentation

7.1.3.1 enum RC::RStr_FloatStyle

The styles in which a floating point number can be formatted.

AUTO == determine the format automatically, FIXED == as 0.0000123, SCI == as 1.23e-5 SCI treats the precision parameter as a count of significant digits. FIXED treats the precision parameter as digits right of the decimal point. AUTO treats the precision parameter as the most significant digits to show, but trims trailing zeros.

7.1.3.2 enum RC::RStr_IntStyle

The styles in which an integral number can be formatted.

DEC == decimal, HEX == hexadecimal, HEX0x == hexadecimal with "0x" prepended, OCT == octal, OCT0 == octal with "0" prepended, BIN == binary, CHAR == treated as a character.

7.1.3.3 enum RC::WriteMode

Valid write modes for a FileWrite or FileRW.

TRUNCATE sets the file size to 0. KEEP preserves the file contents, setting the initial position to the beginning of the file. APPEND starts the read position at the beginning of the file but all Write/Put operations are forced to the end of the file. NEWONLY will only succeed if the file does not already exist.

7.1.4 Function Documentation

7.1.4.1 template<class T > BetwCompare<T> RC::Betw (const T & *x*)

Returns a comparator that can be used for range comparisons.

Usage exmple: if (3.14 < Betw(5) <= 7) { cout << "Yes\n"; } Works for any comparable types, as long as one of the left-most two in the chain is wrapped in Betw.

7.1.4.2 u64 RC::EntropyRND_Get_Range (u64 *range*) [inline]

Uses EntropyRND as a RandomNumberGenerator, e.g. for random_shuffle.

Parameters

range	The range of values that can be returned, starting with 0.

Returns

A random value from 0 up to and including range-1.

7.1.4.3 template<class T > bool RC::FileGetWrapper (FileRead & *fr,* T & *data*) [inline]

Overload this with a specialization to support Get on a custom type.

The default FileGetWrapper for plain old data, which calls RawGet.

An implementation of this should use the FileRead's methods Get and RawGet to read other supported types or primitives, then place the result in data.

Returns

True if the Get succeeded.

7.1.4.4 template<class T > void RC::FilePutWrapper (FileWrite & *fw,* const T & *data*) [inline]

Overload this with a specialization to support Put on a custom type.

The default FilePutWrapper for plain old data, which calls RawPut.

An implementation of this should take the value of data and use the FileWrite's methods Put and RawPut to write other supported types or primitives.

7.1.4.5 template<class T2 > auto RC::IndexOf (const T2 & *cont*) -> RangeHelper<decltype(cont.size())> [inline]

Provides an iterator which dereferences to the indices of cont from [0,cont.size()).

Usage example: for (auto i : IndexOf(cont)) { cout << cont[i]; }

7.1.4.6 template<class T > bool RC::IsIntegerType (const T &x *__attribute__(unused)* = 1) [inline]

True if the type is an integer type.

Use as IsIntegerType<u32>() or u32 x; IsIntegerType(x);

7.1.4.7 template<class T > bool RC::IsSignedType (const T &x *__attribute__(unused)* = 0) [inline]

True if the type is signed.

Use as IsSignedType<u32>() or u32 x; IsSignedType(x);

7.1.4.8 template<class T > T RC::LOW_VAL (const T &x *__attribute__(unused)* = std::numeric_limits<T>::↩ ::lowest()) [inline]

Provide the lowest value held by this type, for which no value is less.

Use as LOW_VAL<f32>() or as f32 x = LOW_VAL(x);

7.1.4.9 template<class Ret , class... Params, class Functor > Caller<Ret, Params...> RC::MakeFunctor (Functor *func*)

A special generator for functors, which requires specifying the return type and arguments.

Template parameters are specified as the return type followed by a list of the arguments.

7.1.4.10 template<class T > T RC::MAX_VAL (const T &x *__attribute__(unused)* = std::numeric_limits<T>::↩ ::max()) [inline]

Provide the maximum value held by this type.

Use as MAX_VAL<u32>() or as u32 x = MAX_VAL(x);

7.1.4.11 template<class T > T RC::MIN_POS (const T &x *__attribute__(unused)* = MIN_POS_↩ Helper<T, FloatType<T>::value>::F()) [inline]

Provide the minimum positive value held by this type.

Use as MIN_POS<f32>() or as f32 x = MIN_POS(x);

7.1.4.12 template<class T > T RC::MIN_VAL (const T &x *__attribute__(unused)* = std::numeric_limits<T>::↩ ::min()) [inline]

Provide the minimum value held by this type. For floats, the smallest positive value.

Use as MIN_VAL<u32>() or as u32 x = MIN_VAL(x);

7.1.4.13 template<class... Args> OneOfCompare<Args...> RC::OneOf (Args... *args*)

Returns a comparator that can be used to check if something is in a list.

Each element given as a parameter to OneOf is constructed with the type of the object left of the == or != operator. Usage exmple: RStr str = "foo"; if (str != OneOf("bar", -3.54, "blah")) { cout << "Not Found\n"; } int x = 8; if (5 == OneOf(3.14, 5, x)) { cout << "Found\n"; }

7.1.4.14 template<class T > std::ostream & RC::operator<< (std::ostream & *out*, const Data2D< T > & *d*) [inline]

Outputs data to a stream as { { x_0_0, x_1_0, ...}, { x_0_1, x_1_1, ...}, ... }.

Usage like: Data2D<RStr> data; std::cout << data << std::endl;

7.1.4.15 template<class T > std::ostream & RC::operator<< (std::ostream & *out,* const Data3D< T > & *d*) [inline]

Outputs data to a stream.

For d_x_y_z, formatted as { { { d_0_0_0, d_1_0_0, ...}, { d_0_1_0, d_1_1_0, ...}, ... }, { { d_0_0_1, d_1_0_1, ...}, { d_0_1_1, d_1_1_1, ...}, ... }, ... } Usage like: Data3D<RStr> data; std::cout << data << std::endl;

7.1.4.16 std::ostream& RC::operator<< (std::ostream & *out,* const Bitfield2D & *b*) [inline]

Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2.

Usage like Bitfield2D bf; std::cout << bf << std::endl;

7.1.4.17 std::ostream& RC::operator<< (std::ostream & *out,* const Bitfield3D & *b*) [inline]

Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2, and double-spaces after entries of dimension 3.

Usage like Bitfield3D bf; std::cout << bf << std::endl;

7.1.4.18 std::ostream& RC::operator<< (std::ostream & *out,* const Bitfield & *b*) [inline]

Outputs data to a stream as a character series of '1's and '0's.

Usage like Bitfield bf; std::cout << bf << std::endl;

7.1.4.19 std::ostream& RC::operator<< (std::ostream & *out,* const Data1D< char > & *d*) [inline]

Outputs data to a stream as { a, b, ... }.

Usage like: Data1D<char> data; std::cout << data << std::endl;

7.1.4.20 std::ostream& RC::operator<< (std::ostream & *out,* const Data1D< u8 > & *d*) [inline]

Outputs data to a stream as { 61, 62, ... }.

Usage like: Data1D<u8> data; std::cout << data << std::endl;

7.1.4.21 std::ostream& RC::operator<< (std::ostream & *out,* const Data1D< i8 > & *d*) [inline]

Outputs data to a stream as { 61, -42, ... }.

Usage like: Data1D<i8> data; std::cout << data << std::endl;

7.1.4.22 template<class T > std::ostream& RC::operator<< (std::ostream & *out,* const Data1D< T > & *d*) [inline]

Outputs data to a stream as { elem0, elem1, ... }.

Usage like: Data1D<RStr> data; std::cout << data << std::endl;

7.1.4.23 template<class T > RangeHelper<T> RC::Range (const T & *start,* const T & *past_the_end*) [inline]

Provides an iterator which dereferences to the iterated values [start,past_the_end).

Usage example, outputs 5 through 9: for (auto i : Range(5, 10)) { cout << i << endl; } This is a generalization, since: for (auto iter : Range(cont.begin(), cont.end())) { ... } is equivalent to the canonical: for (auto iter : cont) { ... }

7.1.4.24 u64 RC::RND_Get_Range (u64 *range*) [inline]

Uses RND as a RandomNumberGenerator, e.g. for random_shuffle.

Parameters

range	The range of values that can be returned, starting with 0.

Returns

A random value from 0 up to and including range-1.

7.1.4.25 u64 RC::URand_Get_Range (u64 *range*) [inline]

Uses URand as a RandomNumberGenerator, e.g. for random_shuffle.

Parameters

range	The range of values that can be returned, starting with 0.

Returns

A random value from 0 up to and including range-1.

Chapter 8

Class Documentation

8.1 RC::APtr< T > Class Template Reference

A reference counting ptr that auto-deletes what it points to when the last copy leaves scope or is otherwise destructed.

`#include <APtr.h>`

Public Member Functions

- APtr (T *t_ptr=NULL)

  Default constructor assigning the value of the pointer.
- APtr (const APtr< T > &other)

  Copy constructor.
- template<class Tderived >
  APtr (const Ptr< Tderived > &other)

  A conversion constructor which creates a new APtr from a Ptr of the same or a derived type.
- APtr & operator= (const APtr< T > &other)

  An assignment operator, which increases the shared reference count with the source APtr.
- ~APtr ()

  Destructor which decreases the shared reference count.
- void Delete ()

  Delete the object pointed to and set the shared pointer for all linked APtr's to NULL.
- T * Raw () const

  Returns a direct reference to the enclosed pointer.
- void Assert () const

  Throws RC::ErrorMsgNull if the pointer is null.
- bool IsSet () const

  True if the pointer is non-NULL.
- bool IsNull () const

  True if the pointer is NULL.
- T & operator* ()

  Dereferences the pointer, or throws an exception if null.
- const T & operator* () const

  Const version of operator()*

- T * operator-> ()

  Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.
- const T * operator-> () const

  Const version of operator->()
- operator T * () const

  Implicitly casts to the enclosed pointer, without null checking.
- template<class Type >
  Ptr< Type > Cast ()

  Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.
- template<class Type >
  const Ptr< Type > Cast () const

  Const version of Cast()
- template<class Type >
  bool CanCast () const

  True if it can dynamically cast to this type.
- template<class Type >
  Type & As ()

  Dynamically casts and dereferences to the type presented as a template parameter, or throws ErrorMsgCast if it fails.
- template<class Type >
  const Type & As () const

  Const version of As()
- T & As ()

  Dereference as the default type.
- const T & As () const

  Const version of operator()*

8.1.1 Detailed Description

template<class T>
class RC::APtr< T >

A reference counting ptr that auto-deletes what it points to when the last copy leaves scope or is otherwise destructed.

Use this for scope-controlled garbage collection. Like Ptr, dereferencing a null APtr throws ErrorMsgNull. For C++11 the reference counting is thread safe.

See also

Ptr
RevPtr

8.1.2 Constructor & Destructor Documentation

8.1.2.1 template<class T> RC::APtr< T >::APtr (T * *t_ptr* = NULL) [inline]

Default constructor assigning the value of the pointer.

Parameters

t_ptr	The new pointer value.

8.1.2.2 template<class T> RC::APtr< T >::APtr (const APtr< T > & *other*) [inline]

Copy constructor.

Parameters

other	The APtr to copy.

8.1.2.3 template<class T> template<class Tderived > RC::APtr< T >::APtr (const Ptr< Tderived > & *other*) [inline]

A conversion constructor which creates a new APtr from a Ptr of the same or a derived type.

Parameters

other	The Ptr from which the pointer should be used to make this APtr.

8.1.2.4 template<class T> RC::APtr< T >::~APtr ( ) [inline]

Destructor which decreases the shared reference count.

The object is deleted if the shared reference count reaches 0.

8.1.3 Member Function Documentation

8.1.3.1 template<class T> T& RC::APtr< T >::As ( ) [inline]

Dereference as the default type.

See also

operator()

8.1.3.2 template<class T> template<class Type > Ptr<Type> RC::APtr< T >::Cast ( ) [inline]

Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.

The Ptr is NULL if the cast fails.

8.1.3.3 template<class T> void RC::APtr< T >::Delete ( ) [inline]

Delete the object pointed to and set the shared pointer for all linked APtr's to NULL.

Note: While this NULLs all the linked APtr's, it cannot affect raw pointers which have already been extracted or are in use when this is called. Be mindful of this in threading architecture design.

8.1.3.4 template<class T> bool RC::APtr< T >::IsNull ( ) const [inline]

True if the pointer is NULL.

Returns

True if the pointer is NULL.

8.1.3.5 template<class T> bool RC::APtr< T >::IsSet ( ) const [inline]

True if the pointer is non-NULL.

Returns

True if the pointer is non-NULL.

8.1.3.6 template<class T> T& RC::APtr< T >::operator* ( ) [inline]

Dereferences the pointer, or throws an exception if null.

The exception thrown is RC::ErrorMsgNull.

Returns

A reference to the dereferenced object.

8.1.3.7 template<class T> T* RC::APtr< T >::operator-> ( ) [inline]

Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.

Returns

The enclosed pointer.

8.1.3.8 template<class T> APtr& RC::APtr< T >::operator= (const APtr< T > & *other*) [inline]

An assignment operator, which increases the shared reference count with the source APtr.

Parameters

other	The source APtr to assign here.

8.1.3.9 template<class T> T* RC::APtr< T >::Raw ( ) const [inline]

Returns a direct reference to the enclosed pointer.

The reference is read/write, so it can be used as a function parameter which updates the value of this pointer.

Returns

A reference to the enclosed pointer.

The documentation for this class was generated from the following file:

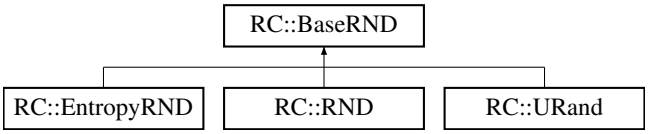
- APtr.h

8.2 RC::BaseRND Class Reference

An abstract class which provides functions for obtaining randomness in convenient forms.

`#include <RND.h>`

Inheritance diagram for RC::BaseRND:



Public Types

- typedef u32 result_type

  For UniformRandomNumberGenerator compliance.

Public Member Functions

- BaseRND ()

  Default constructor.
- virtual u32 Get_u32 ()=0

  Implement this for all subclasses. All other functions get their randomness from this.
- u32 operator() ()

  Equivalent to Get_u32().
- u32 min () const

  Returns 0 for UniformRandomNumberGenerator compliance.
- u32 max () const

  Returns u32 max for UniformRandomNumberGenerator compliance.
- i32 Get_i32 ()

  Provides random i32 values.
- u64 Get_u64 ()

  Provides random u64 values.
- i64 Get_i64 ()

  Provides random i64 values.
- f32 Get_f32 ()

  Provides a random f32 in the range [0,1).
- f64 Get_f64 ()

  Provides a random f64 in the range [0,1).
- u8 Get_u8 ()

  Provides random u8 values.
- i8 Get_i8 ()

  Provides random i8 values.
- char Get_char ()

  Provides random char values.
- u16 Get_u16 ()

  Provides random u16 values.
- i16 Get_i16 ()

  Provides random i16 values.
- void Get (u8 &x)

  Overloaded function to extract type u8 from this class.
- void Get (i8 &x)

  Overloaded function to extract type i8 from this class.
- void Get (char &x)

  Overloaded function to extract type char from this class.
- void Get (u16 &x)

  Overloaded function to extract type u16 from this class.
- void Get (i16 &x)

  Overloaded function to extract type i16 from this class.
- void Get (u32 &x)

  Overloaded function to extract type u32 from this class.
- void Get (i32 &x)

  Overloaded function to extract type i32 from this class.
- void Get (u64 &x)

  Overloaded function to extract type u64 from this class.
- void Get (i64 &x)

  Overloaded function to extract type i64 from this class.
- void Get (f32 &x)

Overloaded function to extract type f32 from this class.

- void Get (f64 &x)

  Overloaded function to extract type f64 from this class.
- bool GetProb (f64 probability)

  Returns true with probability, which should be in the range [0,1].
- bool GetProb (f32 probability)

  Returns true with probability, which should be in the range [0,1].
- u64 GetRange (u64 range)

  Returns [0,range)
- i64 GetRange (i64 range)

  Like GetRange() but for i64.
- u32 GetRange (u32 range)

  Like GetRange() but for i32.
- i32 GetRange (i32 range)

  Like GetRange() but for i32.
- template<class T >
  T GetRange (T low, T high)

  Provides a random integer value in the range [low,high].
- f64 GetFRange (f64 low, f64 high)

  Provides a random float in the range [low,high).
- template<class T >
  void Fill (Data1D< T > &data)

  For any supported data type, fills the Data1D with random values.
- template<class T >
  T GetFrom (Data1D< T > &data)

  Returns a random element from the Data1D.

Static Public Member Functions

- static u64 GetEntropy ()

  Provides 64 bits of environmental entropy.

8.2.1 Detailed Description

An abstract class which provides functions for obtaining randomness in convenient forms.

It fulfills the UniformRandomNumberGenerator concept.

See also

RND
EntropyRND
URand

8.2.2 Member Function Documentation

8.2.2.1 void RC::BaseRND::Get (u8 & *x*) [inline]

Overloaded function to extract type u8 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.2 void RC::BaseRND::Get (i8 & *x*) [inline]

Overloaded function to extract type i8 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.3 void RC::BaseRND::Get (char & *x*) [inline]

Overloaded function to extract type char from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.4 void RC::BaseRND::Get (u16 & *x*) [inline]

Overloaded function to extract type u16 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.5 void RC::BaseRND::Get (i16 & *x*) [inline]

Overloaded function to extract type i16 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.6 void RC::BaseRND::Get (u32 & *x*) [inline]

Overloaded function to extract type u32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.7 void RC::BaseRND::Get (i32 & *x*) [inline]

Overloaded function to extract type i32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.8 void RC::BaseRND::Get (u64 & *x*) [inline]

Overloaded function to extract type u64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.9 void RC::BaseRND::Get (i64 & *x*) [inline]

Overloaded function to extract type i64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.10 void RC::BaseRND::Get (f32 & *x*) [inline]

Overloaded function to extract type f32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.11 void RC::BaseRND::Get (f64 & *x*) [inline]

Overloaded function to extract type f64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.2.2.12 virtual u32 RC::BaseRND::Get_u32 ( ) [pure virtual]

Implement this for all subclasses. All other functions get their randomness from this.

Returns

A random unsigned 32 bit integer.

Implemented in RC::URand, RC::EntropyRND, and RC::RND.

8.2.2.13 static u64 RC::BaseRND::GetEntropy ( ) [inline], [static]

Provides 64 bits of environmental entropy.

This function uses the jittering of a tight loop relative to the ticking of the high precision clock to generate entropy, which is then mixed with the Rijndael S-box. It automatically adapts to the system's clock precision and relative cpu speed. The randomness has passed dieharder version 3.31.1.

Returns

64 bits of environmental entropy.

8.2.2.14 bool RC::BaseRND::GetProb (f64 *probability*) [inline]

Returns true with probability, which should be in the range [0,1].

Parameters

probability	The chance of returning true.

Returns

Randomly true according to probability.

8.2.2.15 bool RC::BaseRND::GetProb (f32 *probability*) [inline]

Returns true with probability, which should be in the range [0,1].

Parameters

probability	The chance of returning true.

Returns

Randomly true according to probability.

8.2.2.16 u64 RC::BaseRND::GetRange (u64 *range*) [inline]

Returns [0,range)

Note: The precision comes from f64, so it is slightly less than full u64 precision.

Parameters

range	The number of integers in the range.

Returns

A value from 0 up to range-1.

8.2.2.17 u32 RC::BaseRND::GetRange (u32 *range*) [inline]

Like GetRange() but for i32.

Note: The precision comes from f32, so it is slightly less than full u32 precision.

The documentation for this class was generated from the following file:

- RND.h

8.3 RC::Bitfield Class Reference

A bounds-safe one-dimensional vector-like structure of efficiently packed bits.

```
#include <Bitfield.h>
```

Classes

- class BitfieldBool

  A temporary return-type that serves as an interface to specific bit values.
- class BitfieldBoolConst

  A temporary return-type that serves as a const interface to specific bit values.

Public Member Functions

- Bitfield ()

  Default constructor which initializes to size 0.
- Bitfield (size_t b_size)

  Constructor which sets the initial size to b_size bits.
- void Delete ()

  Deletes the stored data, resets the size to 0, and releases the memory.
- bool IsEmpty () const

  True if the size is 0.
- BitfieldBool operator[ ] (size_t x)

  Bounds-checked access of the element at index x.
- BitfieldBool operator() (size_t x)

  Identical to operator[].
- BitfieldBoolConst operator[ ] (size_t x) const

  Const version of operator[].
- BitfieldBoolConst operator() (size_t x) const

  Const version of operator[].
- BitfieldBool At (size_t x)

  Const version of At()
- BitfieldBoolConst At (size_t x) const

  Identical to operator[].
- size_t size () const

  Return the current number of bits.
- size_t reserved () const

  Return the number of bits for which space is reserved.
- void Zero ()

  Set all bits to zero.
- void ZeroRange (const size_t start, const size_t end)

  Set all bits to zero between index start and end, inclusive.
- Data1D< u32 > & Raw ()

  Provides raw access to the Data1D<u32> in which the bits are packed.
- void Reserve (const size_t reserve_size)

  Reserve storage without resizing the Bitfield.
- void Resize (const size_t resize_size)

  Resize the array, reallocating if necessary.
- void Crop ()

  Reduce memory consumption to only that necessary for size() bits.
- void Clear ()

  Identical to Delete()
- void Append (const bool new_bit)

  Add a bit to the end, expanding if necessary.
- void Append (const Bitfield &other)

  Append bits from another Bitfield to this Bitfield.
- void ExpandSet (size_t pos, const bool new_bit)

  Assign new_bit to index pos, expanding if necessary to reach that index.
- Bitfield & operator+= (const bool new_bit)

  Appends new_bit, expanding as necessary.
- Bitfield & operator+= (const Bitfield &other)

  Appends the bits from other, expanding as necessary.
- bool Check (const size_t x) const

True if index x is within the Bitfield.

- void Assert (const size_t x) const

  Throws ErrorMsgBounds if index x is out of bounds for this Bitfield.

- size_t CountOnes () const

  Efficiently determines the total number of true / 1 values in the Bitfield.

- size_t CountZeroes () const

  Efficiently determines the total number of false / 0 values in the Bitfield.

Friends

- class **Bitfield2D**
- class **Bitfield3D**
- void swap (Bitfield &a, Bitfield &b)

  Swap the data in Bitfield a and b.

8.3.1 Detailed Description

A bounds-safe one-dimensional vector-like structure of efficiently packed bits.

It provides efficient resizing and append operations, and bounds-safe array access.

See also

Bitfield2D
Bitfield3D

8.3.2 Constructor & Destructor Documentation

8.3.2.1 RC::Bitfield::Bitfield (size_t *b_size*) [inline], [explicit]

Constructor which sets the initial size to b_size bits.

Parameters

b_size	The initial number of bits.

8.3.3 Member Function Documentation

8.3.3.1 void RC::Bitfield::Append (const bool *new_bit*) [inline]

Add a bit to the end, expanding if necessary.

Efficiency note: Expands by doubling for linear efficiency.

Parameters

new_bit	The new bit to append to the end.

See also

Reserve

8.3.3.2 void RC::Bitfield::Append (const Bitfield & *other*) [inline]

Append bits from another Bitfield to this Bitfield.

Parameters

other	The Bitfield from which bits should be copied over.

8.3.3.3 Bitfield& RC::Bitfield::operator+= (const bool *new_bit*) [inline]

Appends new_bit, expanding as necessary.

See also

Append

8.3.3.4 Bitfield& RC::Bitfield::operator+= (const Bitfield & *other*) [inline]

Appends the bits from other, expanding as necessary.

See also

Append

8.3.3.5 BitfieldBool RC::Bitfield::operator[ ] (size_t *x*) [inline]

Bounds-checked access of the element at index x.

Throws an ErrorMsgBounds exception if x is out of the bounds.

Parameters

x	Index of the bit to access.

Returns

A BitfieldBool structure that casts to a bool or can be assigned a bool value.

8.3.3.6 Data1D<u32>& RC::Bitfield::Raw ( ) [inline]

Provides raw access to the Data1D<u32> in which the bits are packed.

The bits are packed in order, least significant bit first.

8.3.3.7 void RC::Bitfield::Reserve (const size_t *reserve_size*) [inline]

Reserve storage without resizing the Bitfield.

Parameters

reserve_size	The number of bits for which storage should be reserved.

8.3.3.8 void RC::Bitfield::Resize (const size_t *resize_size*) [inline]

Resize the array, reallocating if necessary.

See Data1D::Resize for efficiency details.

The documentation for this class was generated from the following file:

- Bitfield.h

8.4 RC::Bitfield2D Class Reference

A bounds-safe two-dimensional structure of efficiently packed bits.

`#include <Bitfield2D.h>`

Public Member Functions

- Bitfield2D ()
  Default constructor which initializes to size 0, 0.
- Bitfield2D (size_t b_size1, size_t b_size2)
  Constructor which sets the initial sizes.
- void Delete ()
  Deletes the stored data, resets the sizes to 0, and releases the memory.
- bool IsEmpty () const
  True if the total size of the bits is 0.
- Bitfield::BitfieldBool operator() (size_t x, size_t y)
  Bounds-checked access of the indexed element.
- Bitfield::BitfieldBoolConst operator() (size_t x, size_t y) const
  Const version of operator()
- Bitfield::BitfieldBool At (size_t x, size_t y)
  Identical to operator()
- Bitfield::BitfieldBoolConst At (size_t x, size_t y) const
  Const version of At()
- size_t size1 () const
  Return the extent in bits of dimension 1.

- size_t size2 () const

  Return the extent in bits of dimension 2.
- void Zero ()

  Set all bits to 0 / false.
- void Zero1D (const size_t y)

  Set all bits of dimension 1 to zero at y indexed dimension 2.
- bool Check (const size_t x, const size_t y) const

  True if indices x and y are both in bounds.
- void Assert (const size_t x, const size_t y) const

  Throws ErrorMsgBounds if index x or y is out of bounds.

Friends

- void swap (Bitfield2D &a, Bitfield2D &b)

  Swap the data in Bitfield2D a and b.

8.4.1 Detailed Description

A bounds-safe two-dimensional structure of efficiently packed bits.

Note AxB and BxA dimensions will both take AB bits of storage with the total rounded up to the nearest 32 bit word.

See also

Bitfield
Bitfield3D

8.4.2 Constructor & Destructor Documentation

8.4.2.1 RC::Bitfield2D::Bitfield2D (size_t *b_size1*, size_t *b_size2*) [inline], [explicit]

Constructor which sets the initial sizes.

Efficiency note: The first dimension, set by b_size1, should be the one which is iterated over most frequently in innermost loops for caching gains.

Parameters

b_size1	The size of the first dimension.
b_size2	The size of the second dimension.

8.4.3 Member Function Documentation

8.4.3.1 Bitfield::BitfieldBool RC::Bitfield2D::operator() (size_t *x*, size_t *y*) [inline]

Bounds-checked access of the indexed element.

Throws an ErrorMsgBounds exception if x or y is out of the bounds.

Parameters

x	The dimension 1 index of the bit to access.
y	The dimension 2 index of the bit to access.

Returns

A Bitfield::BitfieldBool structure that casts to a bool or can be assigned a bool value.

The documentation for this class was generated from the following file:

- Bitfield2D.h

8.5 RC::Bitfield3D Class Reference

A bounds-safe three-dimensional structure of efficiently packed bits.

`#include <Bitfield3D.h>`

Public Member Functions

- Bitfield3D ()
  Default constructor which initializes to size 0, 0, 0.
- Bitfield3D (size_t b_size1, size_t b_size2, size_t b_size3)
  Cosntructor which sets the initial sizes.
- void Delete ()
  Deletes the stored data, resets the sizes to 0, and releases the memory.
- bool IsEmpty () const
  True of the total size of the bits is 0.
- Bitfield::BitfieldBool operator() (size_t x, size_t y, size_t z)
  Bounds-checked access of the indexed element.
- Bitfield::BitfieldBoolConst operator() (size_t x, size_t y, size_t z) const
  Const version of operator()
- Bitfield::BitfieldBool At (size_t x, size_t y, size_t z)
  Identical to operator()
- Bitfield::BitfieldBoolConst At (size_t x, size_t y, size_t z) const
  Const version of At()
- size_t size1 () const
  Return the extent in bits of dimension 1.
- size_t size2 () const
  Return the extent in bits of dimension 2.
- size_t size3 () const
  Return the extent in bits of dimension 3.
- void Zero ()
  Set all bits to 0 / false.
- void Zero2D (const size_t z)
  Set all bits of dimensions 1 and 2 to zero at z indexed dimension 3.
- void Zero1D (const size_t y, const size_t z)
  Set all bits of dimension 1 to zero at y and z indexed dimensions 2 and 3.
- bool Check (const size_t x, const size_t y, const size_t z) const
  True of indices x, y, and z are all in bounds.
- void Assert (const size_t x, const size_t y, const size_t z) const
  Throws ErrorMsgBounds if index x, y, or z is out of bounds.

Friends

- void swap (Bitfield3D &a, Bitfield3D &b)

  Swap the data in Bitfield3D a and b.

8.5.1 Detailed Description

A bounds-safe three-dimensional structure of efficiently packed bits.

Note AxBxC dimensions take ABC bits of storage with the total rounded up to the nearest 32 bit word.

See also

Bitfield
Bitfield2D

8.5.2 Constructor & Destructor Documentation

8.5.2.1 RC::Bitfield3D::Bitfield3D (size_t *b_size1*, size_t *b_size2*, size_t *b_size3*) [inline], [explicit]

Cosntructor which sets the initial sizes.

Efficiency note: the first dimension, set by b_size1, should be the one which is iterated over most frequently in innermost loops for caching gains.

Parameters

b_size1	The size of the first dimension.
b_size2	The size of the second dimension.
b_size3	The size of the third dimension.

8.5.3 Member Function Documentation

8.5.3.1 Bitfield::BitfieldBool RC::Bitfield3D::operator() (size_t *x*, size_t *y*, size_t *z*) [inline]

Bounds-checked access of the indexed element.

Throws an ErrorMsgBounds exception if x, y, or z is out of the bounds.

Parameters

x	The dimension 1 index of the bit to access.
y	The dimension 2 index of the bit to access.
z	The dimension 3 index of the bit to access.

Returns

A Bitfield::BitfieldBool structure that casts to a bool or can be assigned a bool value.

The documentation for this class was generated from the following file:

- Bitfield3D.h

8.6 RC::Bitfield::BitfieldBool Class Reference

A temporary return-type that serves as an interface to specific bit values.

`#include <Bitfield.h>`

Public Member Functions

- operator bool () const
  Implicit conversion to bool.
- void Set ()
  Set the bit to true / 1.
- void Clear ()
  Set the bit to false / 0.
- BitfieldBool & operator= (const bool set)
  Assign the bool value of set to this bit.

Friends

- class **Bitfield**

8.6.1 Detailed Description

A temporary return-type that serves as an interface to specific bit values.

The documentation for this class was generated from the following file:

- Bitfield.h

8.7 RC::Bitfield::BitfieldBoolConst Class Reference

A temporary return-type that serves as a const interface to specific bit values.

`#include <Bitfield.h>`

Public Member Functions

- operator bool () const

  Implicit conversion to bool.

Friends

- class **Bitfield**

8.7.1 Detailed Description

A temporary return-type that serves as a const interface to specific bit values.

The documentation for this class was generated from the following file:

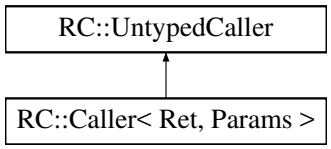
- Bitfield.h

8.8 RC::Caller< Ret, Params > Class Template Reference

A general purpose function class which can refer to any static method, member method, functor, or lambda function.

`#include <Caller.h>`

Inheritance diagram for RC::Caller< Ret, Params >:



Public Member Functions

- Caller ()

  Default constructor, refers to no function.
- Caller (const Caller &other)

  Copy constructor.
- template<class C >
  Caller (C *object, Ret(C::*func)(Params...))

  Construct a Caller to a member function of a specific object.
- template<class C >
  Caller (C &object, Ret(C::*func)(Params...))

  Construct a Caller to a member function of a specific object.
- template<class C , class... OneFewerParams>
  Caller (Ret(C::*func)(OneFewerParams...))

  Construct a Caller to a member function, where the inserted first parameter is a reference to the object.
- template<class Functor >
  Caller (Functor func)

Construct a Caller to any general functor.

- virtual ~Caller ()

  Destructor.

- Caller & operator= (const Caller< Ret, Params... > &other)

  Assign a Caller of identical type.

- virtual Ret operator() (Params...params) const

  Call the referenced function.

- bool IsSet ()

  True if a function was set.

- template<class TupleType >
  Ret Use (TupleType tup)

  Call the referenced function with the parameters given as RC::Tuple tup.

- template<class... Args>
  auto Bind (Args...args) -> decltype(std::bind(*this, args...))

  Return a functor with arguments bound using syntax identical to std::bind.

8.8.1 Detailed Description

template<class Ret, class... Params>
class RC::Caller< Ret, Params >

A general purpose function class which can refer to any static method, member method, functor, or lambda function.

Template parameters are specified as the return type followed by a list of the argument types. If access is attempted on an undefined Caller, ErrorMsgNull is thrown. If a member method Caller is constructed with the object as the first parameter, then the call is assigned to that object, which can be used for passing handlers. The convenience functions MakeCaller and MakeFunctor can be used for automatic type inference. Strict compile time type-checking is performed upon assignment or construction. To explicitly wrap with loose type-checking, use MakeFunctor on a Caller.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 template<class Ret, class... Params> template<class C , class... OneFewerParams> RC::Caller< Ret, Params >::Caller (Ret(C::*)(OneFewerParams...) *func*) [inline]

Construct a Caller to a member function, where the inserted first parameter is a reference to the object.

Usage example: class A { void F(int x) {} }; Caller<void, A&, int> c(&A::F); c(5);

8.8.2.2 template<class Ret, class... Params> template<class Functor > RC::Caller< Ret, Params >::Caller (Functor *func*) [inline]

Construct a Caller to any general functor.

Note, for a Caller functor of a slightly different type, use MakeFunctor to avoid the intentional type-checking compile error.

8.8.3 Member Function Documentation

8.8.3.1 template<class Ret, class... Params> template<class... Args> auto RC::Caller< Ret, Params >::Bind (Args... *args*) -> decltype(std::bind(*this, args...)) [inline]

Return a functor with arguments bound using syntax identical to std::bind.

The return type of this is an unspecified functor, but it can be wrapped in a MakeFunctor with the corresponding types.

The documentation for this class was generated from the following file:

- Caller.h

8.9 RC::Data1D< T > Class Template Reference

A bounds-safe one-dimensional vector-like structure.

`#include <Data1D.h>`

Public Member Functions

- Data1D (size_t d_size)
  Constructor which sets the initial size to d_size.
- Data1D (const Data1D< T > &copy)
  Copy constructor that copies all elements.
- Data1D (const std::initializer_list< T > &new_data)
  Initializer list constructor, for initializing with bracketed data.
- Data1D (Data1D< T > &&other)
  Move constructor that reassigns all the data to this Data1D.
- Data1D (size_t d_size, T *new_data, bool auto_delete=false)
  Efficiently wraps in-place C-pointer data in a bounds-safe container.
- void Delete ()
  Delete all the elements and free all allocated memory.
- ~Data1D ()
  Deletes all contents upon destruction.
- bool IsEmpty () const
  Returns true if there are no elements / the size is 0.
- template<class T2 >
  size_t Find (const T2 &elem, size_t start_at=0) const
  Returns the first index at or after start_at for which the data equals elem.
- template<class T2 >
  bool Contains (const T2 &elem) const
  Returns true if at least one entry equals elem.
- bool Check (const size_t x) const
  Check if index x is in bounds.
- void Assert (const size_t x) const
  Throws an ErrorMsgBounds exception if x is out of bounds.
- void Reserve (const size_t reserve_size)

Reserve storage without resizing the array.

- void Resize (const size_t resize_size)

  Resize the array, reallocating if necessary.
- void SetOffset (const size_t new_offset)

  Set a new offset position for index 0.
- void SetRange (const size_t new_offset, const size_t new_size)

  Set a new offset and data size, resizing if necessary.
- T * Raw () const

  Access a raw unprotected C-pointer to the enclosed data.
- size_t size () const

  Returns the current number of elements.
- size_t TypeSize () const

  Returns the size of this Data1D's type.
- size_t ByteSize () const

  Returns the current size in bytes.
- size_t reserved () const

  Returns the number of elements for which space is reserved.
- size_t ByteReserved () const

  Returns the current allocated storage in bytes.
- size_t GetOffset () const

  Returns the current offset for index 0.
- Data1D Copy (const size_t pos=0, const size_t amnt=npos) const

  Creates a duplicate copy of the contents, with up to amnt elements from pos.
- template<class T2 >
  Data1D & CopyFrom (const Data1D< T2 > &other)

  Copy data from any type with a compatible assignment operator.
- template<class T2 >
  Data1D & CopyFrom (const Data1D< T2 > &other, size_t pos, size_t num_elem=npos)

  assignment operator.
- template<class T2 >
  void CopyAt (const size_t pos, const Data1D< T2 > &other)

  Overwrite a range of data using all data from any compatible type, expanding if necessary.
- template<class T2 >
  void CopyAt (const size_t pos, const Data1D< T2 > &other, const size_t other_start, const size_t amnt=npos)

  Overwrite a range of data from any compatible type, expanding if necessary.
- void CopyData (const size_t dest, const size_t source, const size_t amnt=npos)

  Copy data from any location in this Data1D to another location, handling overlap automatically.
- Data1D & operator= (const Data1D &other)

  Assignment operator which copies all contents from other, respecting offsets.
- Data1D & operator= (Data1D &&other)

  Assignment operator which copies all contents from other, respecting offsets.
- template<class T2 >
  Data1D< T2 > Cast ()

  Returns a new array with all the elements of this array assigned to type T2.
- template<class Conv >
  auto CastWith (Conv converter) -> Data1D< decltype(converter(T

  Returns a new array with all the elements of this array converted to another type by converter.

Static Public Attributes

- static const size_t npos = size_t(-1)

  The largest possible value of size_t.

8.9.1 Detailed Description

template<class T>
class RC::Data1D< T >

A bounds-safe one-dimensional vector-like structure.

It provides efficient resizing and offsets, as well as convenience functions for assignments and comparisons. It also provides bounds-safe iterators. Note: Non-POD classes stored in Data1D containers must have a default constructor with default values or no arguments, which is necessary for efficient resizing. This requirement also permits safe use of offsets and temporary shrinking, because all allocated space contains defined data with properly constructed objects.

See also

Data2D
Data3D

8.9.2 Constructor & Destructor Documentation

8.9.2.1 template<class T> RC::Data1D< T >::Data1D (size_t *d_size*) [inline], [explicit]

Constructor which sets the initial size to d_size.

Parameters

d_size	The initial number of elements.

8.9.2.2 template<class T> RC::Data1D< T >::Data1D (const Data1D< T > & *copy*) [inline]

Copy constructor that copies all elements.

Parameters

copy	A source Data1D from which elements should be copied.

8.9.2.3 template<class T> RC::Data1D< T >::Data1D (const std::initializer_list< T > & *new_data*) [inline]

Initializer list constructor, for initializing with bracketed data.

Parameters

new_data	An initialization list of the form {elem1, elem2, ...}.

8.9.2.4 template<class T> RC::Data1D< T >::Data1D (Data1D< T > && *other*) [inline]

Move constructor that reassigns all the data to this Data1D.

Parameters

other	A source Data1D from which elements should be moved.

8.9.2.5 template<class T> RC::Data1D< T >::Data1D (size_t *d_size*, T * *new_data*, bool *auto_delete* = false) [inline]

Efficiently wraps in-place C-pointer data in a bounds-safe container.

Note: If auto_delete=true, new_data must have been acquired with new[]. delete[] is used internally, so behavior may vary if it was acquired with new, malloc, or others.

Parameters

d_size	The initial number of elements.
new_data	C-pointer data to wrap.
auto_delete	Should the Data1D delete new_data upon destruction?

8.9.2.6 template<class T> RC::Data1D< T >::~Data1D ( ) [inline]

Deletes all contents upon destruction.

Note: For wrapped pointers, if auto_delete was false no deletion of the original data occurs.

8.9.3 Member Function Documentation

8.9.3.1 template<class T> void RC::Data1D< T >::Assert (const size_t *x*) const [inline]

Throws an ErrorMsgBounds exception if x is out of bounds.

Parameters

x	The index to check.

8.9.3.2 template<class T> template<class T2 > Data1D<T2> RC::Data1D< T >::Cast ( ) [inline]

Returns a new array with all the elements of this array assigned to type T2.

See also

CopyFrom

8.9.3.3 template<class T> bool RC::Data1D< T >::Check (const size_t *x*) const [inline]

Check if index x is in bounds.

Parameters

x	The index to check.

Returns

True if in bounds.

8.9.3.4 template<class T> template<class T2 > void RC::Data1D< T >::CopyAt (const size_t *pos,* const Data1D< T2 > & *other*) [inline]

Overwrite a range of data using all data from any compatible type, expanding if necessary.

Parameters

pos	The offset in this Data1D to begin overwriting.
other	The compatible Data1D to copy from.

8.9.3.5 template<class T> template<class T2 > void RC::Data1D< T >::CopyAt (const size_t *pos,* const Data1D< T2 > & *other,* const size_t *other_start,* const size_t *amnt* = npos) [inline]

Overwrite a range of data from any compatible type, expanding if necessary.

Parameters

pos	The offset in this Data1D to begin overwriting.
other	The compatible Data1D to copy from.
other_start	The offset in the other Data1D to begin copying from.
amnt	The quantity of data to copy (automatically bounds-capped, defaults to all).

8.9.3.6 template<class T> void RC::Data1D< T >::CopyData (const size_t *dest,* const size_t *source,* const size_t *amnt* = npos) [inline]

Copy data from any location in this Data1D to another location, handling overlap automatically.

Parameters

dest	The offset in this Data1D to begin overwriting.
source	The offset in this Data1D to copy from.
amnt	The quantity of data to copy (automatically bounds-capped, defaults to all).

8.9.3.7 template<class T> template<class T2 > Data1D& RC::Data1D< T >::CopyFrom (const Data1D< T2 > & *other*) [inline]

Copy data from any type with a compatible assignment operator.

Parameters

other	The compatible Data1D from which data should be copied.

Returns

A reference to this Data1D.

8.9.3.8 template<class T> template<class T2 > Data1D& RC::Data1D< T >::CopyFrom (const Data1D< T2 > & *other,* size_t *pos,* size_t *num_elem* = npos) [inline]

assignment operator.

Note, to copy data around in the same Data1D, use CopyData.

Parameters

other	The compatible Data1D from which data should be copied.
pos	The offset in other to start copying from.
num_elem	The number of elements to copy.

Returns

A reference to this Data1D.

8.9.3.9 template<class T> template<class T2 > size_t RC::Data1D< T >::Find (const T2 & *elem,* size_t *start_at* = 0) const [inline]

Returns the first index at or after start_at for which the data equals elem.

Parameters

elem	The element to compare to each entry in this Data1D
start↩ _at	The first index to begin comparing at.

Returns

The index of the item found, or npos if no matching data is found.

8.9.3.10 template<class T> size_t RC::Data1D< T >::GetOffset ( ) const [inline]

Returns the current offset for index 0.

See also

SetOffset
SetRange

8.9.3.11 template<class T> Data1D& RC::Data1D< T >::operator= (const Data1D< T > & *other*) [inline]

Assignment operator which copies all contents from other, respecting offsets.

Parameters

other	Data1D to copy from, from offset (default 0) to size.

Returns

This object.

8.9.3.12 template<class T> Data1D& RC::Data1D< T >::operator= (Data1D< T > && *other*) [inline]

Assignment operator which copies all contents from other, respecting offsets.

Parameters

other	Data1D to copy from, from offset (default 0) to size.

Returns

This object.

8.9.3.13 template<class T> T* RC::Data1D< T >::Raw ( ) const [inline]

Access a raw unprotected C-pointer to the enclosed data.

Warning: This convenience function bypasses the bounds protections provided by this class. The returned pointer is likely to become invalid after any of the operations on this object which change the size.

Returns

C-style pointer to the contents at the offset (0 by default).

8.9.3.14 template<class T> void RC::Data1D< T >::Reserve (const size_t *reserve_size*) [inline]

Reserve storage without resizing the array.

Parameters

reserve_size	The number of elements worth of storage to reserve.

8.9.3.15 template<class T> void RC::Data1D< T >::Resize (const size_t *resize_size*) [inline]

Resize the array, reallocating if necessary.

This may trigger a copy operation upon expansion. For efficiency it never reallocates or copies while shrinking or expanding within a previous size range. Use Crop if necessary to shrink storage to the current size.

Parameters

resize_size	The new number of elements.

8.9.3.16 template<class T> void RC::Data1D< T >::SetOffset (const size_t *new_offset*) [inline]

Set a new offset position for index 0.

This also adjusts the size to reach the same last element.

Parameters

new_offset	The new absolute offset from the 0th allocated element.

See also

SetRange

8.9.3.17 template<class T> void RC::Data1D< T >::SetRange (const size_t *new_offset*, const size_t *new_size*) [inline]

Set a new offset and data size, resizing if necessary.

Parameters

new_offset	The new absolute offset from the 0th allocated element.
new_size	The new number of elements.

See also

SetOffset
Resize

The documentation for this class was generated from the following file:

- Data1D.h

8.10 RC::Data2D< T > Class Template Reference

A bounds-safe two-dimensional resizeable structure.

`#include <Data2D.h>`

Public Member Functions

- Data2D ()
  Default constructor which initializes to size 0.
- Data2D (size_t d_size1, size_t d_size2)
  Constructor which sets the initial sizes.
- Data2D (const Data2D< T > &copy)
  Copy constructor that copies all elements.
- Data2D (const std::initializer_list< Data1D< T >> &new_data)
  Initializer list constructor, initializes with nested brackets.
- ~Data2D ()
  Deletes all contents upon destruction.
- void Delete ()
  Delete all the elements and free all allocated memory.
- void Clear ()
  Identical to Delete().
- void Crop ()
  Reduces memory consumption to only that necessary for the current size.
- bool IsEmpty () const
- Data2D & operator= (const Data2D &other)
  Assignment operator which copies all contents from other.
- RAIter< Data1D< Data1D< T > >, Data1D< T > > begin ()
  Copy data from any other object of a type with a compatible.
- const RAIter< Data1D< Data1D< T > >, Data1D< T > > begin () const
  Const version of begin.
- RAIter< Data1D< Data1D< T > >, Data1D< T > > end ()
  Return a bounds-checked random-access iterator starting just past the last element.
- RAIter< Data1D< Data1D< T > >, Data1D< T > > end () const
  Const version end.
- Data1D< T > & operator[ ] (size_t x)
  Bounds-checked access of a Data1D corresponding to the data at index x in dimension 2.
- Data1D< T > & operator() (size_t x)
  Identical to Data2D::operator[ ].
- const Data1D< T > & operator[ ] (size_t x) const
  Const version of Data2D::operator[ ].
- const Data1D< T > & operator() (size_t x) const
  Const version of Data2D::operator[ ].
- T & operator() (size_t x, size_t y)
  Bounds-checked access of an element.
- const T & operator() (size_t x, size_t y) const
  Const version of Data2D::operator()(size_t x, size_t y)
- T & At (size_t x, size_t y)
  Equivalent to Data2D::operator()(size_t x, size_t y)
- const T & At (size_t x, size_t y) const

Const version of Data2D::operator()(size_t x, size_t y)

- size_t size1 () const

  Get the size of dimension 1.
- size_t size2 () const

  Get the size of dimension 2.
- size_t TypeSize () const

  Returns sizeof(T).
- void Zero ()

  Sets all elements equal to 0.
- bool Check (const size_t x, const size_t y) const

  Check if the indices x and y are in bounds.
- void Assert (const size_t x, const size_t y) const

  Throw an ErrorMsgBounds exception if either x or y is out of bounds.
- void Resize (const size_t resize_size1, const size_t resize_size2)

  Resize the array, reallocating if necessary.
- T ** Raw ()

  Access a raw unprotected 2-dimensional C-array for the enclosed data.
- Data1D< Data1D< T > > & RawData ()

  Access the underlying nested Data1D structure for this object.
- const Data1D< Data1D< T > > & RawData () const

  Const version of RawData().
- void ToLilEndian ()

  Convert endianness of all elements if needed, for supported types.
- void FromLilEndian ()

  Convert endianness of all elements if needed, for supported types.
- void ToBigEndian ()

  Convert endianness of all elements if needed, for supported types.
- void FromBigEndian ()

  Convert endianness of all elements if needed, for supported types.

Friends

- template<class T2 >
  void swap (Data2D< T2 > &a, Data2D< T2 > &b)

  Efficiently swap all the contents of a and b.

8.10.1 Detailed Description

template<class T>
class RC::Data2D< T >

A bounds-safe two-dimensional resizeable structure.

Note: Non-POD classes stored in Data2D containers must have a default constructor with default values or no arguments.

See also

Data1D
Data3D

8.10.2 Constructor & Destructor Documentation

8.10.2.1 template<class T> RC::Data2D< T >::Data2D (size_t *d_size1*, size_t *d_size2*) [inline], [explicit]

Constructor which sets the initial sizes.

Efficiency note: The first dimension, set by d_size1, should be the one which is iterated over most frequently in innermost loops for caching gains.

Parameters

d_size1	The size of the first dimension.
d_size2	The size of the second dimension.

8.10.2.2 template<class T> RC::Data2D< T >::Data2D (const Data2D< T > & *copy*) [inline]

Copy constructor that copies all elements.

Parameters

copy	A source Data2D from which elements should be copied.

8.10.3 Member Function Documentation

8.10.3.1 template<class T> void RC::Data2D< T >::Assert (const size_t *x*, const size_t *y*) const [inline]

Throw an ErrorMsgBounds exception if either x or y is out of bounds.

Parameters

x	The index for dimension 1.
y	The index for dimension 2.

8.10.3.2 template<class T> RAIter< Data1D< Data1D<T> >, Data1D<T> > RC::Data2D< T >::begin ( ) [inline]

Copy data from any other object of a type with a compatible.

Return a bounds-checked random-access iterator starting at offset. Note that the iterator is revokable, meaning use of it will throw an exception if this Data2D is deleted.

Returns

A bounds-checked iterator.

See also

end()

8.10.3.3 template<class T> bool RC::Data2D< T >::Check (const size_t *x*, const size_t *y*) const [inline]

Check if the indices x and y are in bounds.

Parameters

x	The index for dimension 1.
y	The index for dimension 2.

Returns

True if in bounds.

8.10.3.4 template<class T> void RC::Data2D< T >::Crop ( ) [inline]

Reduces memory consumption to only that necessary for the current size.

Efficiency note: This function may copy all elements if necessary for reallocating.

8.10.3.5 template<class T> RAIter< Data1D< Data1D<T> >, Data1D<T> > RC::Data2D< T >::end ( ) [inline]

Return a bounds-checked random-access iterator starting just past the last element.

Note that the iterator is revokable, meaning use of it will throw an exception if this Data2D is deleted.

Returns

A bounds-checked iterator.

See also

begin()

8.10.3.6 template<class T> bool RC::Data2D< T >::IsEmpty ( ) const [inline]

Returns

True if there are no elements / at least one size is 0.

8.10.3.7 template<class T> T& RC::Data2D< T >::operator() (size_t *x*, size_t *y*) [inline]

Bounds-checked access of an element.

Throws an ErrorMsgBounds if out of bounds. Note that for Data2D<int> arr; arr(x, y) is equivalent to arr[y][x].

Parameters

x	The index of dimension 1
y	The index of dimension 2.

Returns

A reference to the indexed element.

8.10.3.8 template<class T> Data2D& RC::Data2D< T >::operator= (const Data2D< T > & *other*) [inline]

Assignment operator which copies all contents from other.

Parameters

other	Data2D to copy from.

Returns

This object.

8.10.3.9 template<class T> Data1D<T>& RC::Data2D< T >::operator[ ] (size_t *x*) [inline]

Bounds-checked access of a Data1D corresponding to the data at index x in dimension 2.

Throws an ErrorMsgBounds if out of bounds. Usage note: Data2D<int> arr(size1, size2); int val = arr[x2][x1];

Parameters

x	The index of dimension 2.

Returns

A Data1D of all elements in dimension 1 at that index.

8.10.3.10 template<class T> T** RC::Data2D< T >::Raw ( ) [inline]

Access a raw unprotected 2-dimensional C-array for the enclosed data.

Warning: This convenience function bypasses the bounds protections provided by this class. Also the C-array becomes invalid if this object is resized, deleted, or cropped.

Returns

C-style pointer to the contents.

8.10.3.11 template<class T> Data1D< Data1D<T> >& RC::Data2D< T >::RawData ( ) [inline]

Access the underlying nested Data1D structure for this object.

Attempts should not be made to resize the underlying data accessed with this convenience function.

Returns

The nested Data1D contained within.

8.10.3.12 template<class T> void RC::Data2D< T >::Resize (const size_t *resize_size1*, const size_t *resize_size2*) [inline]

Resize the array, reallocating if necessary.

This may trigger a copy operation upon expansion. For efficiency, it never reallocates or copies while shrinking or expanding within a previous size range. Use Crop if necessary to shrink storage to the current size.

Parameters

resize_size1	The new size for dimension 1.
resize_size2	The new size for dimension 2.

8.10.3.13 template<class T> void RC::Data2D< T >::Zero ( ) [inline]

Sets all elements equal to 0.

See also

Data1D::Zero()

The documentation for this class was generated from the following file:

- Data2D.h

8.11 RC::Data3D< T > Class Template Reference

A bounds-safe three-dimensional resizeable structure.

```
#include <Data3D.h>
```

Public Member Functions

- Data3D ()

  Default constructor which initializes to size 0, 0, 0.
- Data3D (size_t d_size1, size_t d_size2, size_t d_size3)

  Constructor which sets the initial sizes.
- Data3D (const Data3D< T > &copy)

  Copy constructor that copies all elements.
- ~Data3D ()

  Deletes all contents upon destruction.
- void Delete ()

  Delete all the elements and free all allocated memory.
- void Clear ()

  Identical to Delete().
- void Crop ()

  Reduces memory consumption to only that necessary for the current size.
- bool IsEmpty () const
- Data3D & operator= (const Data3D &other)

  Assignment operator which copies all contents from other.
- Data2D< T > & operator[ ] (size_t x)

  Bounds-checked access of a Data2D corresponding to the data at index x in dimension 3.
- Data2D< T > & operator() (size_t x)

  Identical to Data3D::operator[ ].
- const Data2D< T > & operator[ ] (size_t x) const

  Const version of Data3D::operator[ ].
- const Data2D< T > & operator() (size_t x) const

  Const version of Data3D::operator[ ].
- T & operator() (size_t x, size_t y, size_t z)

  Bounds-checked access of an element.
- const T & operator() (size_t x, size_t y, size_t z) const

  Const version of Data3D::operator()(size_t x, size_t y, size_t z)
- T & At (size_t x, size_t y, size_t z)

  Equivalent to Data3D::operator()(size_t x, size_t y, size_t z)
- const T & At (size_t x, size_t y, size_t z) const

  Const version of Data3D::operator()(size_t x, size_t y, size_t z))
- size_t size1 () const

  Get the size of dimension 1.
- size_t size2 () const

  Get the size of dimension 2.
- size_t size3 () const

  Get the size of dimension 3.
- size_t TypeSize () const

  Returns sizeof(T).
- void Zero ()

  Sets all elements equal to 0.
- bool Check (const size_t x, const size_t y, const size_t z) const

  Check if the indices x, y, and z are in bounds.
- void Assert (const size_t x, const size_t y, const size_t z) const

  Throw an ErrorMsgBounds exception if either x, y, or z is out of bounds.
- void Resize (const size_t resize_size1, const size_t resize_size2, const size_t resize_size3)

  Resize the array, reallocating if necessary.

- T *** Raw ()

  Access a raw unprotected 3-dimensional C-array for the enclosed data.
- const Data1D< Data2D< T > > & RawData () const

  Access the underlying nested Data1D/Data2D structure for this object.
- Data1D< Data2D< T > > & RawData ()

  Const version of RawData().
- void ToLilEndian ()

  Convert endianness of all elements if needed, for supported types.
- void FromLilEndian ()

  Convert endianness of all elements if needed, for supported types.
- void ToBigEndian ()

  Convert endianness of all elements if needed, for supported types.
- void FromBigEndian ()

  Convert endianness of all elements if needed, for supported types.

Friends

- template<class T2 >
  void swap (Data3D< T2 > &a, Data3D< T2 > &b)

  Efficiently swap all the contents of a and b.

8.11.1 Detailed Description

template<class T>
class RC::Data3D< T >

A bounds-safe three-dimensional resizeable structure.

See also

Data1D
Data2D

8.11.2 Constructor & Destructor Documentation

8.11.2.1 template<class T> RC::Data3D< T >::Data3D (size_t *d_size1*, size_t *d_size2*, size_t *d_size3*) [inline], [explicit]

Constructor which sets the initial sizes.

Efficiency note: The first dimension, set by d_size1, should be the one which is iterated over most frequently in innermost loops for caching gains. Note: Non-POD classes stored in Data3D containers must have a default constructor with default values or no arguments.

Parameters

d_size1	The size of the first dimension.
d_size2	The size of the second dimension.
d_size3	The size of the third dimension.

8.11.2.2 template<class T> RC::Data3D< T >::Data3D (const Data3D< T > & *copy*) [inline]

Copy constructor that copies all elements.

Parameters

copy	A source Data3D from which elements should be copied.

8.11.3 Member Function Documentation

8.11.3.1 template<class T> void RC::Data3D< T >::Assert (const size_t *x*, const size_t *y*, const size_t *z*) const [inline]

Throw an ErrorMsgBounds exception if either x, y, or z is out of bounds.

Parameters

x	The index for dimension 1.
y	The index for dimension 2.
z	The index for dimension 3.

8.11.3.2 template<class T> bool RC::Data3D< T >::Check (const size_t *x*, const size_t *y*, const size_t *z*) const [inline]

Check if the indices x, y, and z are in bounds.

Parameters

x	The index for dimension 1.
y	The index for dimension 2.
z	The index for dimension 3.

Returns

True of in bounds.

8.11.3.3 template<class T> void RC::Data3D< T >::Crop ( ) [inline]

Reduces memory consumption to only that necessary for the current size.

Efficiency note: This function may copy all elements if necessary for reallocating.

8.11.3.4 template<class T> bool RC::Data3D< T >::IsEmpty ( ) const [inline]

Returns

True of there are no elements / at least one size is 0.

8.11.3.5 template<class T> T& RC::Data3D< T >::operator() (size_t *x*, size_t *y*, size_t *z*) [inline]

Bounds-checked access of an element.

Throws an ErrorMsgBounds if out of bounds. Note that for Data3D<int> arr; arr(x, y, z) is equivalent to arr[z][y][x].

Parameters

x	The index of dimension 1
y	The index of dimension 2.
z	The index of dimension 3.

Returns

A reference to the indexed element.

8.11.3.6 template<class T> Data3D& RC::Data3D< T >::operator= (const Data3D< T > & *other*) [inline]

Assignment operator which copies all contents from other.

Parameters

other	Data3D to copy from.

Returns

This object.

8.11.3.7 template<class T> Data2D<T>& RC::Data3D< T >::operator[ ] (size_t *x*) [inline]

Bounds-checked access of a Data2D corresponding to the data at index x in dimension 3.

Throws an ErrorMsgBounds if out of bounds. Usage note: Data3D<int> arr(size1, size2, size3); int val = arr[x3][x2][x1];

Parameters

x	The index of dimension 3.

Returns

A Data2D of the elements in dimension 1 and 2 at that index.

8.11.3.8 template<class T> T*** RC::Data3D< T >::Raw ( ) [inline]

Access a raw unprotected 3-dimensional C-array for the enclosed data.

Warning: This convenience function bypasses the bounds protections provided by this class.

Returns

C-style pointer to the contents.

8.11.3.9 template<class T> const Data1D< Data2D<T> >& RC::Data3D< T >::RawData ( ) const [inline]

Access the underlying nested Data1D/Data2D structure for this object.

Attempts should not be made to resize the underlying data accessed with this convenience function.

Returns

The nested Data1D/Data2D contained within.

8.11.3.10 template<class T> void RC::Data3D< T >::Resize (const size_t *resize_size1*, const size_t *resize_size2*, const size_t *resize_size3*) [inline]

Resize the array, reallocating if necessary.

This may trigger a copy operation upon expansion. For efficiency, it never reallocates or copies while shrinking or expanding within a previous size range. Use Crop if necessary to shrink storage to the current size.

Parameters

resize_size1	The new size for dimension 1.
resize_size2	The new size for dimension 2.
resize_size3	The new size for dimension 2.

8.11.3.11 template<class T> void RC::Data3D< T >::Zero ( ) [inline]

Sets all elements equal to 0.

See also

Data1D::Zero()

The documentation for this class was generated from the following file:

- Data3D.h

8.12 RC::DebugTrack< ClassTracking, stack_trace > Class Template Reference

Inherit this class to add construction, destruction, and assignment output tracking.

`#include <RCBits.h>`

8.12.1 Detailed Description

template<class ClassTracking, bool stack_trace = false>
class RC::DebugTrack< ClassTracking, stack_trace >

Inherit this class to add construction, destruction, and assignment output tracking.

Set ClassTracking to the class which is inheriting this. If stack_trace is true, a stack trace is output at each event. See ErrorMsg for stack trace usage details.

The documentation for this class was generated from the following file:

- RCBits.h

8.13 RC::DynCaller Class Reference

A typeless container for a Caller, which has methods for dynamically casting it to the correct type.

`#include <Caller.h>`

Public Member Functions

- DynCaller ()
  Default constructor, contains no Caller.
- template<class... Types>
  DynCaller (Caller< Types... > new_caller)
  Construct a DynCaller which wraps new_caller.
- template<class Ret , class... Params>
  DynCaller (Ret(*func)(Params...))
  Creates a Caller referring to the given static function, and then wraps it.
- template<class... Types>
  Caller< Types... > & As ()
  Dynamically casts to the Caller of the specified template parameters, or throws ErrorMsgCast if it fails.
- template<class... Types>
  const Caller< Types... > & As () const
  Const version of As().
- template<class... Types>
  bool CanCast () const
  True if this can cast to a Caller with the specified template parameters.

8.13.1 Detailed Description

A typeless container for a Caller, which has methods for dynamically casting it to the correct type.

The documentation for this class was generated from the following file:

- Caller.h

8.14 RC::Endian Class Reference

Auto-detects the endianness of the compilation target, and provides automatic endian conversion features.

`#include <Types.h>`

Static Public Member Functions

- static u16 Swap (u16 x)
  Reverse the byte order.
- static u32 Swap (u32 x)
  Reverse the byte order.
- static u64 Swap (u64 x)
  Reverse the byte order.
- static u8 Swap (u8 x)
  Do nothing. (For consistency.)
- static i8 Swap (i8 x)
  Do nothing. (For consistency.)
- static i16 Swap (i16 x)
  Reverse the byte order.
- static i32 Swap (i32 x)
  Reverse the byte order.
- static i64 Swap (i64 x)
  Reverse the byte order.
- template<class T >
  static T ToLittle (const T &x)
  Converts x to little-endian.
- template<class T >
  static T FromLittle (const T &x)
  Converts x from little-endian.
- template<class T >
  static T ToBig (const T &x)
  Converts x to big-endian.
- template<class T >
  static T FromBig (const T &x)
  Converts x from big-endian.
- static bool IsLittle ()
  True if this system is little-endian.
- static bool IsBig ()
  True if this system is big-endian.

8.14.1 Detailed Description

Auto-detects the endianness of the compilation target, and provides automatic endian conversion features.

The documentation for this class was generated from the following file:

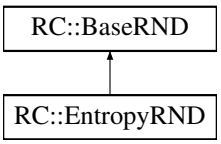
- Types.h

8.15 RC::EntropyRND Class Reference

Provides true random numbers sourced from environmental noise.

`#include <RND.h>`

Inheritance diagram for RC::EntropyRND:



Public Member Functions

- EntropyRND ()
  Default constructor.
- virtual u32 Get_u32 ()
  Provides random u32 values.

Additional Inherited Members

8.15.1 Detailed Description

Provides true random numbers sourced from environmental noise.

See the entropy source BaseRND::GetEntropy() for details on the method. These numbers should be cryptographically strong.

The documentation for this class was generated from the following file:

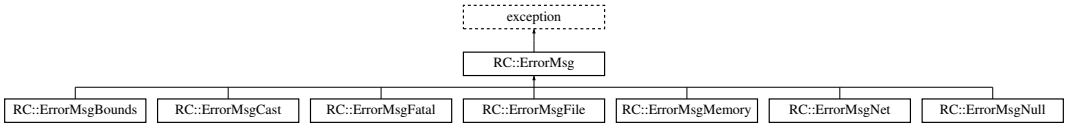
- RND.h

8.16 RC::ErrorMsg Class Reference

An exception class that records where the exception was thrown and provides a stack trace.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsg:



Public Member Functions

- ErrorMsg (const char *new_err_msg=NULL, const char *filename=NULL, int line_number=0) noexcept
  The default constructor.
- virtual ~ErrorMsg () noexcept
  The destructor.
- ErrorMsg (const ErrorMsg &)=default
  Default copy constructor.
- ErrorMsg (ErrorMsg &&)=default
  Default move constructor.
- ErrorMsg & operator= (const ErrorMsg &)&=default
  Default copy assignment operator.
- ErrorMsg & operator= (ErrorMsg &&)&=default
  Default move assignment operator.
- virtual const char * GetError () const noexcept
  Provides the reason given for the error.
- virtual const char * GetType () const noexcept
  Provides the type of the exception if this is a subclass.
- virtual bool IsError (const char *test_err) const noexcept
  Return true if the the reason for the exception matches test_err.
- virtual const char * what () const noexcept
  Returns a full descriptive error message with reason, source location, and stack trace.

8.16.1 Detailed Description

An exception class that records where the exception was thrown and provides a stack trace.

Use with the macro Throw_RC_Error("Reason"); to automatically pick up the source filename and line number. This exception should be used liberally any time an error would be informative for resolving a bug.

Stack tracing is implemented for Linux and Windows. For Linux with g++ or clang compile with -rdynamic for full symbols in the stack trace output from what(). For Windows symbols cannot be automatically provided with mingw, but the addresses provided in what() can be resolved by providing them as stdin to "addr2line -pfe myprogram.exe" for any program compiled with -ggdb.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 RC::ErrorMsg::ErrorMsg (const char * *new_err_msg* = NULL, const char * *filename* = NULL, int *line_number* = 0) [inline], [noexcept]

The default constructor.

Use the convenience macro Throw_RC_Error("Reason"); which uses **FILE** and **LINE** to extract the location of the throw.

Parameters

new_err_msg	The reason for the exception.
filename	The source code file that the exception was triggered in.
line_number	The source code line number that the exception was triggered on.

8.16.3 Member Function Documentation

8.16.3.1 virtual const char* RC::ErrorMsg::GetError ( ) const [inline], [virtual], [noexcept]

Provides the reason given for the error.

Returns

The reason for the exception being thrown.

8.16.3.2 virtual const char* RC::ErrorMsg::GetType ( ) const [inline], [virtual], [noexcept]

Provides the type of the exception if this is a subclass.

An empty string is returned for the base class.

Returns

The exception type defined in a subclass.

8.16.3.3 virtual bool RC::ErrorMsg::IsError (const char * *test_err*) const [inline], [virtual], [noexcept]

Return true if the the reason for the exception matches test_err.

Parameters

test_err	An error message to compare with the reason given in the constructor.

Returns

True if the error messages are identical.

8.16.3.4 virtual const char* RC::ErrorMsg::what ( ) const [inline], [virtual], [noexcept]

Returns a full descriptive error message with reason, source location, and stack trace.

Note: The stack trace is only available if on a supported system, and if the RC_NO_STACKTRACE option was not given in RCconfig.h

The documentation for this class was generated from the following file:

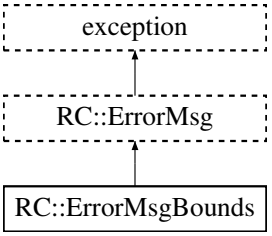
- Errors.h

8.17 RC::ErrorMsgBounds Class Reference

A subtype of RC::ErrorMsg for Bounds errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgBounds:



Public Member Functions

- ErrorMsgBounds (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgBounds () noexcept
  The destructor.
- ErrorMsgBounds (const ErrorMsgBounds &)=default
  Default copy constructor.
- ErrorMsgBounds (ErrorMsgBounds &&)=default
  Default move constructor.
- ErrorMsgBounds & operator= (const ErrorMsgBounds &)&=default
  Default copy assignment operator.
- ErrorMsgBounds & operator= (ErrorMsgBounds &&)&=default
  Default move assignment operator.

8.17.1 Detailed Description

A subtype of RC::ErrorMsg for Bounds errors.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 RC::ErrorMsgBounds::ErrorMsgBounds (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Bounds , "Reason");

The documentation for this class was generated from the following file:

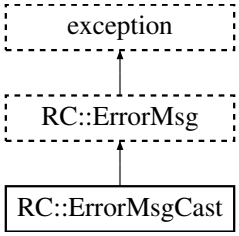
- Errors.h

8.18 RC::ErrorMsgCast Class Reference

A subtype of RC::ErrorMsg for Bad Cast errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgCast:



Public Member Functions

- ErrorMsgCast (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgCast () noexcept
  The destructor.
- ErrorMsgCast (const ErrorMsgCast &)=default
  Default copy constructor.
- ErrorMsgCast (ErrorMsgCast &&)=default
  Default move constructor.
- ErrorMsgCast & operator= (const ErrorMsgCast &)&=default
  Default copy assignment operator.
- ErrorMsgCast & operator= (ErrorMsgCast &&)&=default
  Default move assignment operator.

8.18.1 Detailed Description

A subtype of RC::ErrorMsg for Bad Cast errors.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 RC::ErrorMsgCast::ErrorMsgCast (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Cast , "Reason");

The documentation for this class was generated from the following file:

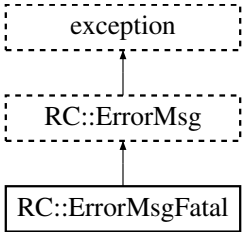
- Errors.h

8.19 RC::ErrorMsgFatal Class Reference

A subtype of RC::ErrorMsg for Fatal errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgFatal:



Public Member Functions

- ErrorMsgFatal (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgFatal () noexcept
  The destructor.
- ErrorMsgFatal (const ErrorMsgFatal &)=default
  Default copy constructor.
- ErrorMsgFatal (ErrorMsgFatal &&)=default
  Default move constructor.
- ErrorMsgFatal & operator= (const ErrorMsgFatal &)&=default
  Default copy assignment operator.
- ErrorMsgFatal & operator= (ErrorMsgFatal &&)&=default
  Default move assignment operator.

8.19.1 Detailed Description

A subtype of RC::ErrorMsg for Fatal errors.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 RC::ErrorMsgFatal::ErrorMsgFatal (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Fatal , "Reason");

The documentation for this class was generated from the following file:

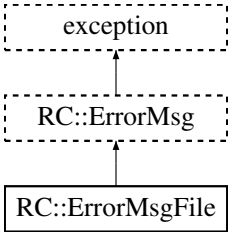
- Errors.h

8.20 RC::ErrorMsgFile Class Reference

A subtype of RC::ErrorMsg for File related errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgFile:



Public Member Functions

- ErrorMsgFile (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgFile () noexcept
  The destructor.
- ErrorMsgFile (const ErrorMsgFile &)=default
  Default copy constructor.
- ErrorMsgFile (ErrorMsgFile &&)=default
  Default move constructor.
- ErrorMsgFile & operator= (const ErrorMsgFile &)&=default
  Default copy assignment operator.
- ErrorMsgFile & operator= (ErrorMsgFile &&)&=default
  Default move assignment operator.

8.20.1 Detailed Description

A subtype of RC::ErrorMsg for File related errors.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 RC::ErrorMsgFile::ErrorMsgFile (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(File , "Reason");

The documentation for this class was generated from the following file:

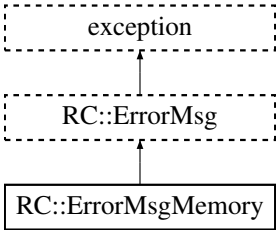
- Errors.h

8.21 RC::ErrorMsgMemory Class Reference

A subtype of RC::ErrorMsg for Memory errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgMemory:



Public Member Functions

- ErrorMsgMemory (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgMemory () noexcept
  The destructor.
- ErrorMsgMemory (const ErrorMsgMemory &)=default
  Default copy constructor.
- ErrorMsgMemory (ErrorMsgMemory &&)=default
  Default move constructor.
- ErrorMsgMemory & operator= (const ErrorMsgMemory &)&=default
  Default copy assignment operator.
- ErrorMsgMemory & operator= (ErrorMsgMemory &&)&=default
  Default move assignment operator.

8.21.1 Detailed Description

A subtype of RC::ErrorMsg for Memory errors.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 RC::ErrorMsgMemory::ErrorMsgMemory (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Memory , "Reason");

The documentation for this class was generated from the following file:

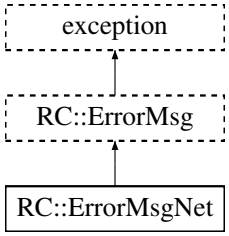
- Errors.h

8.22 RC::ErrorMsgNet Class Reference

A subtype of RC::ErrorMsg for Networking related errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgNet:



Public Member Functions

- ErrorMsgNet (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgNet () noexcept
  The destructor.
- ErrorMsgNet (const ErrorMsgNet &)=default
  Default copy constructor.
- ErrorMsgNet (ErrorMsgNet &&)=default
  Default move constructor.
- ErrorMsgNet & operator= (const ErrorMsgNet &)&=default
  Default copy assignment operator.
- ErrorMsgNet & operator= (ErrorMsgNet &&)&=default
  Default move assignment operator.

8.22.1 Detailed Description

A subtype of RC::ErrorMsg for Networking related errors.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 RC::ErrorMsgNet::ErrorMsgNet (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Net , "Reason");

The documentation for this class was generated from the following file:

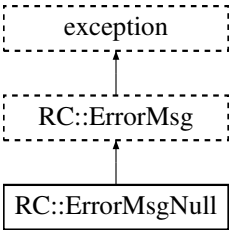
- Errors.h

8.23 RC::ErrorMsgNull Class Reference

A subtype of RC::ErrorMsg for Null errors.

`#include <Errors.h>`

Inheritance diagram for RC::ErrorMsgNull:



Public Member Functions

- ErrorMsgNull (const char *new_err_msg, const char *filename="", int line_number=0)
  The default constructor.
- virtual ~ErrorMsgNull () noexcept
  The destructor.
- ErrorMsgNull (const ErrorMsgNull &)=default
  Default copy constructor.
- ErrorMsgNull (ErrorMsgNull &&)=default
  Default move constructor.
- ErrorMsgNull & operator= (const ErrorMsgNull &)&=default
  Default copy assignment operator.
- ErrorMsgNull & operator= (ErrorMsgNull &&)&=default
  Default move assignment operator.

8.23.1 Detailed Description

A subtype of RC::ErrorMsg for Null errors.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 RC::ErrorMsgNull::ErrorMsgNull (const char * *new_err_msg*, const char * *filename* = " ", int *line_number* = 0) [inline]

The default constructor.

Use the convenience macro Throw_RC_Type(Null , "Reason");

The documentation for this class was generated from the following file:

- Errors.h

8.24 RC::File Class Reference

A class with static methods for file and directory info and manipulation.

`#include <File.h>`

Static Public Member Functions

- static size_t Size (const RStr &pathname)

  Returns the file size of pathname, or 0 if the file does not exist.
- static bool Exists (const RStr &pathname)

  Returns true if the file pathname exists.
- static bool Delete (const RStr &pathname, bool quiet_fail=true)

  Deletes the file pathname, returning true if it succeeded.
- static void Copy (const RStr &srcfile, const RStr &destfile, bool overwrite=true)

  Copies the contents of srcfile to destfile. It will overwrite an existing file only if overwrite is true.
- static void Move (const RStr &srcfile, const RStr &destfile, bool overwrite=true)

  Moves srcfile to destfile by copying the contents and then deleting srcfile.
- static bool MakeDir (const RStr &dirname, bool return_true_if_exists=true)

  Makes a new directory dirname.
- static RStr CurrentDir ()

  Returns the current working directory.
- static Data1D< RStr > DirList (const RStr &path, bool qualified=false)

  Returns a list of entries in the directory path.
- static RStr Dirname (const RStr &filename)

  Extracts the directory portion of filename, or the current directory if filename has no directory.
- static RStr Basename (const RStr &filename)

  Extracts the basename portion of filename, which excludes the directory.
- static RStr FullPath (const RStr &path, const RStr &filename)

  Merges path and filename into a path with a divider inserted if needed.
- static RStr Extension (const RStr &filename)

  Extracts the filename's extension.
- static RStr NoExtension (const RStr &filename)

  Extracts the filename without the extension.

Static Public Attributes

- static const char divider = '\\'

  The OS-specific divider between directories in a pathname.

8.24.1 Detailed Description

A class with static methods for file and directory info and manipulation.

8.24.2 Member Function Documentation

8.24.2.1 static void RC::File::Copy (const RStr & *srcfile,* const RStr & *destfile,* bool *overwrite* = `true`) [inline], [static]

Copies the contents of srcfile to destfile. It will overwrite an existing file only if overwrite is true.

Throws ErrorMsgFile if an error occurs.

8.24.2.2 static bool RC::File::Delete (const RStr & *pathname,* bool *quiet_fail* = `true`) [inline], [static]

Deletes the file pathname, returning true if it succeeded.

If quiet_fail is false, an exception is thrown upon failure.

8.24.2.3 static Data1D<RStr> RC::File::DirList (const RStr & *path,* bool *qualified* = `false`) [inline], [static]

Returns a list of entries in the directory path.

If qualified is true, the entries are fully qualified pathnames. Throws ErrorMsgFile if the directory cannot be read.

8.24.2.4 static bool RC::File::MakeDir (const RStr & *dirname,* bool *return_true_if_exists* = `true`) [inline], [static]

Makes a new directory dirname.

Returns

True if it succeeded. If return_true_if_exists is true, success is if the directory exists upon return. If it's false, success if if the directory was newly created.

8.24.2.5 static void RC::File::Move (const RStr & *srcfile,* const RStr & *destfile,* bool *overwrite* = `true`) [inline], [static]

Moves srcfile to destfile by copying the contents and then deleting srcfile.

Note: This first attempts an efficient rename, but if that fails it falls back to a copy/delete operation. ErrorMsgFile is thrown if the move fails.

The documentation for this class was generated from the following file:

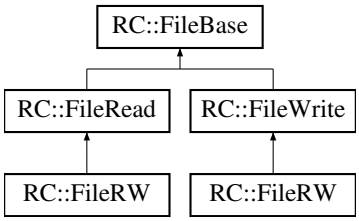
- File.h

8.25 RC::FileBase Class Reference

Provides the common methods for the FileRead/FileWrite/FileRW classes.

`#include <File.h>`

Inheritance diagram for RC::FileBase:



Public Member Functions

- FileBase ()
  Default constructor.
- FileBase (const FileBase &other)
  Copy constructor.
- FileBase & operator= (const FileBase &other)
  Assignment operator.
- virtual ~FileBase ()
  Flushes the write buffer before destructing. This will close the file if it is the last FileBase sharing it.
- RStr GetFilename () const
  Returns the filename if one was given upon opening.
- void SetFilename (const RStr &newfilename) const
  Manually changes the associated filename.
- void Close ()
  Flushes the buffers and closes the file.
- bool IsOpen () const
  True if the file is open.
- bool IsClosed () const
  True if the file is closed.
- bool IsReadable () const
  True if the file is open and readable.
- bool IsWritable () const
  True if the file is open and writable.
- FILE * Raw () const
  Returns a raw FILE which can be used with the C file functions.*
- void Assert () const
  Throws ErrorMsgFile if the file is closed.
- size_t Size () const
  Returns the file size in bytes.
- void SetPosition (const size_t pos)
  Sets the reading and writing position to pos bytes into the file.
- void RelativePosition (const i64 amnt)
  Sets the position for the next read or write operation.

- size_t GetPosition (bool quiet_fail=false) const

  Gets the position for the next read or write operation.
- void Rewind ()

  Clears the read-ahead buffer used by FileRead::Get calls.
- void Flush ()

  Flushes all unsaved data to the storage system.
- void ClearBuffer ()

  Processes the remaining data in the Put/Get buffer.

8.25.1 Detailed Description

Provides the common methods for the FileRead/FileWrite/FileRW classes.

8.25.2 Member Function Documentation

8.25.2.1 size_t RC::FileBase::GetPosition (bool *quiet_fail* = `false`) const [inline]

Gets the position for the next read or write operation.

Parameters

quiet_fail	If true, does not throw an error from being unable to find the position. (0 is returned.)

8.25.2.2 void RC::FileBase::RelativePosition (const i64 *amnt*) [inline]

Sets the position for the next read or write operation.

Note: This has no effect on write operations in APPEND mode.

The documentation for this class was generated from the following file:

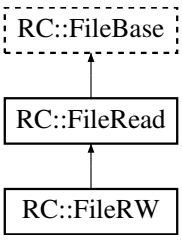
- File.h

8.26 RC::FileRead Class Reference

A file reading class that provides buffered and unbuffered access to files with support for non-POD classes.

`#include <File.h>`

Inheritance diagram for RC::FileRead:



Public Member Functions

- FileRead ()

  Default constructor.
- FileRead (const RStr &filename)

  Opens the file specified by filename for reading.
- FileRead (FILE *fp, bool do_close)

  Wraps the FILE fp for reading. It will close it when finished if do_close is true.*
- FileRead (FILE *fp)

  Wraps the FILE fp for reading. It will close it when finished unless fp is stdin.*
- FileRead (const FileBase &other)

  Base copy constructor.
- bool Open (const RStr &filename)

  Opens the file specified by filename for reading.
- void Open (FILE *fp, bool do_close)

  Wraps the FILE fp for reading. It will close it when finished if do_close is true.*
- void Open (FILE *fp)

  Wraps the FILE fp for reading. It will close it when finished unless fp is stdin.*
- template<class T >
  size_t Read (Data1D< T > &data, const size_t amnt_to_read)

  Reads amnt_to_read elements of plain old data type T into data without buffering, resizing if too small.
- template<class T >
  size_t Read (Data1D< T > &data)

  Fills data with plain old data type T without buffering.
- template<class T >
  void ReadAll (Data1D< T > &data)

  Reads all data until the end of file into data as plain old data type T.
- bool ReadLine (RStr &line, bool crop_newline=true)

  Reads one line until newline, null, or end of file. The newline is removed if crop_newline is true.
- bool Read (RStr &line, bool crop_newline=true)

  Does ReadLine.
- bool SkipLine ()

  Discards one line until newline, null, or end of file.
- void ReadAllLines (Data1D< RStr > &lines, bool crop_newlines=true)

  Reads all the lines found until the end of the file. If crop_newlines is true they are removed from each line.
- void ReadAll (Data1D< RStr > &lines, bool crop_newlines=true)

  Does ReadAllLines.
- template<class T >
  bool RawGet (T &data)

  Performs a buffered read of sizeof(T) bytes and assigns them to data.
- template<class T >
  bool Get (T &data)

  Gets data of type T, after passing through FileGetWrapper.
- bool Get (RStr &data, bool crop_newline=true)

  Gets one line into data, up to the newline, null, or end of file, removing the newline if crop_newline is true.
- bool Get (Data1D< RStr > &data, bool crop_newlines=true)

  Fill data with lines from the file, removing the newlines if crop_newlines is true.
- template<class T >
  bool Get (Data1D< T > &data)

  Fill data with elements of type T, calling Get on each one.
- template<class T >
  bool Get (Data2D< T > &data)

Fill data with elements of type T, calling Get on each one.

- template<class T >
  bool Get (Data3D< T > &data)

  Fill data with elements of type T, calling Get on each one.
- bool GetAll (Data1D< RStr > &data, bool crop_newlines=true)

  Fill data with all lines from the file until the end, removing the newlines if crop_newlines is true.
- template<class T >
  bool GetAll (Data1D< T > &data)

  Fill data with all elements of type T until the end of file, calling Get on each one.

8.26.1 Detailed Description

A file reading class that provides buffered and unbuffered access to files with support for non-POD classes.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 RC::FileRead::FileRead (const RStr & *filename*) [inline]

Opens the file specified by filename for reading.

Throws ErrorMsgFile if the file could not be opened.

8.26.3 Member Function Documentation

8.26.3.1 template<class T > bool RC::FileRead::Get (T & *data*) [inline]

Gets data of type T, after passing through FileGetWrapper.

Returns

True if the read succeeded.

8.26.3.2 bool RC::FileRead::Get (RStr & *data,* bool *crop_newline* = true) [inline]

Gets one line into data, up to the newline, null, or end of file, removing the newline if crop_newline is true.

Returns

True if the read succeeded.

8.26.3.3 bool RC::FileRead::Get (Data1D< RStr > & *data,* bool *crop_newlines* = true) [inline]

Fill data with lines from the file, removing the newlines if crop_newlines is true.

Returns

True if the reads succeeded.

8.26.3.4 template<class T > bool RC::FileRead::Get (Data1D< T > & *data*) [inline]

Fill data with elements of type T, calling Get on each one.

Returns

True if the reads succeeded.

8.26.3.5 template<class T > bool RC::FileRead::Get (Data2D< T > & *data*) [inline]

Fill data with elements of type T, calling Get on each one.

For data[y][x], x is the inner loop.

Returns

True if the reads succeeded.

8.26.3.6 template<class T > bool RC::FileRead::Get (Data3D< T > & *data*) [inline]

Fill data with elements of type T, calling Get on each one.

For data[z][y][x], x is the inner loop.

Returns

True if the reads succeeded.

8.26.3.7 bool RC::FileRead::GetAll (Data1D< RStr > & *data,* bool *crop_newlines* = true) [inline]

Fill data with all lines from the file until the end, removing the newlines if crop_newlines is true.

Returns

True if the reads succeeded.

8.26.3.8 template<class T > bool RC::FileRead::GetAll (Data1D< T > & *data*) [inline]

Fill data with all elements of type T until the end of file, calling Get on each one.

Returns

True if the reads succeeded.

8.26.3.9 bool RC::FileRead::Open (const RStr & *filename*) [inline]

Opens the file specified by filename for reading.

Returns

True if the file was successfully opened.

8.26.3.10 template<class T > bool RC::FileRead::RawGet (T & *data*) [inline]

Performs a buffered read of sizeof(T) bytes and assigns them to data.

Returns

True if the read succeeded.

8.26.3.11 template<class T > size_t RC::FileRead::Read (Data1D< T > & *data,* const size_t *amnt_to_read*) [inline]

Reads amnt_to_read elements of plain old data type T into data without buffering, resizing if too small.

Throws ErrorMsgFile if there was an error reading.

Returns

The number of elements successfully read.

8.26.3.12 template<class T > size_t RC::FileRead::Read (Data1D< T > & *data*) [inline]

Fills data with plain old data type T without buffering.

Throws ErrorMsgFile if there was an error reading.

Returns

The number of elements successfully read.

8.26.3.13 bool RC::FileRead::ReadLine (RStr & *line,* bool *crop_newline* = true) [inline]

Reads one line until newline, null, or end of file. The newline is removed if crop_newline is true.

Throws ErrorMsgFile if there was a read error.

Returns

True if the read was successful.

The documentation for this class was generated from the following file:

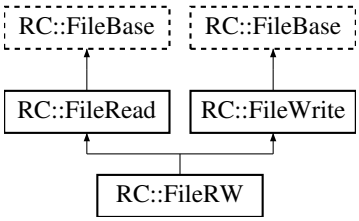
- File.h

8.27 RC::FileRW Class Reference

A file class for both reading and writing that provides buffered and unbuffered output to files.

`#include <File.h>`

Inheritance diagram for RC::FileRW:



Public Member Functions

- FileRW ()

  Default constructor.
- FileRW (const RStr &filename, const WriteMode mode=KEEP)

  Opens the file specified by filename for reading and writing, using the WriteMode specified by mode.
- FileRW (FILE ∗fp, bool do_close)

  Wraps the FILE∗ fp for reading and writing. It will close it when finished if do_close is true.
- FileRW (FILE ∗fp)

  Wraps the FILE∗ fp for reading and writing. It will close it when finished unless fp is stdin/stdout/stderr.
- FileRW (const FileBase &other)

  Base copy constructor.
- bool Open (const RStr &filename, const WriteMode mode=KEEP)

  Opens the file specified by filename for reading and writing, using the WriteMode specified by mode.
- void Open (FILE ∗fp, bool do_close)

  Wraps the FILE∗ fp for reading and writing. It will close it when finished if do_close is true.
- void Open (FILE ∗fp)

  Wraps the FILE∗ fp for reading and writing. It will close it when finished unless fp is stdin/stdout/stderr.

8.27.1 Detailed Description

A file class for both reading and writing that provides buffered and unbuffered output to files.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 RC::FileRW::FileRW (const RStr & *filename,* const WriteMode *mode* = `KEEP`) `[inline]`

Opens the file specified by filename for reading and writing, using the WriteMode specified by mode.

Throws ErrorMsgFile if the file could not be opened.

8.27.3 Member Function Documentation

8.27.3.1 bool RC::FileRW::Open (const RStr & *filename,* const WriteMode *mode* = KEEP) [inline]

Opens the file specified by filename for reading and writing, using the WriteMode specified by mode.

Returns

True if the file was successfully opened.

The documentation for this class was generated from the following file:

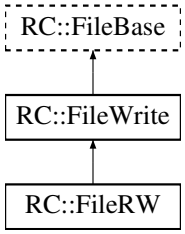
- File.h

8.28 RC::FileWrite Class Reference

A file writing class that provides buffered and unbuffered output to files with support for non-POD classes.

`#include <File.h>`

Inheritance diagram for RC::FileWrite:



Public Member Functions

- FileWrite ()
  Default constructor.
- FileWrite (const RStr &filename, const WriteMode mode=TRUNCATE)
  Opens the file specified by filename for writing, using the WriteMode specified by mode.
- FileWrite (FILE *fp, bool do_close)
  Wraps the FILE fp for writing. It will close it when finished if do_close is true.*
- FileWrite (FILE *fp)
  Wraps the FILE fp for writing. It will close it when finished unless fp is stdout.*
- FileWrite (const FileBase &other)
  Base copy constructor.
- bool Open (const RStr &filename, const WriteMode mode=TRUNCATE)
  Opens the file specified by filename for writing, using the WriteMode specified by mode.
- void Open (FILE *fp, bool do_close)
  Wraps the FILE fp for writing. It will close it when finished if do_close is true.*
- void Open (FILE *fp)
  Wraps the FILE fp for writing. It will close it when finished unless fp is stdout or stderr.*

- template<class T >
  void Write (const Data1D< T > &data, const size_t amnt_to_write)

  Writes amnt_to_write elements of plain old data type T from data without buffering.
- template<class T >
  void Write (const Data1D< T > &data)

  Writes all of data's plain old data type T without buffering.
- void WriteStr (const char *str)

  Writes the null-terminated contents of str without buffering.
- void WriteStr (const RStr &str)

  Writes the RStr str without buffering.
- void WriteAllStr (const Data1D< RStr > &lines, const bool add_newlines)

  Writes all lines without buffering, adding a newline after each if add_newlines is true.
- template<class T >
  void RawPut (const T &data)

  Performs a buffered write of sizeof(T) raw bytes extracted from data.
- template<class T >
  void Put (const T &data)

  Puts data of type T into the write buffer, after passing through FilePutWrapper.
- void Put (const char *str)

  Puts the null-terminated character data str into the write buffer, excluding the null.
- void Put (const RStr &str)

  Puts the RStr str into the write buffer.
- template<class T >
  void Put (const Data1D< T > &data)

  Calls Put for each element of data.
- template<class T >
  void Put (const Data2D< T > &data)

  Calls Put for each element of data.
- template<class T >
  void Put (const Data3D< T > &data)

  Calls Put for each element of data.
- void Put (const Data1D< RStr > &data, const bool add_newline)

  Puts each RStr in data into the write buffer, adding newlines to each if add_newline is true.
- FileWrite & operator<< (FileRead &read)

  Get all the remaining data from the FileRead, and Put it to this FileWrite.

8.28.1 Detailed Description

A file writing class that provides buffered and unbuffered output to files with support for non-POD classes.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 RC::FileWrite::FileWrite (const RStr & *filename*, const WriteMode *mode* = TRUNCATE) [inline]

Opens the file specified by filename for writing, using the WriteMode specified by mode.

Throws ErrorMsgFile if the file could not be opened.

8.28.3 Member Function Documentation

8.28.3.1 bool RC::FileWrite::Open (const RStr & *filename,* const WriteMode *mode* = TRUNCATE) [inline]

Opens the file specified by filename for writing, using the WriteMode specified by mode.

Returns

True if the file was successfully opened.

8.28.3.2 template<class T > void RC::FileWrite::Put (const Data2D< T > & *data*) [inline]

Calls Put for each element of data.

For data[y][x], x is the inner loop.

8.28.3.3 template<class T > void RC::FileWrite::Put (const Data3D< T > & *data*) [inline]

Calls Put for each element of data.

For data[z][y][x], x is the inner loop.

8.28.3.4 template<class T > void RC::FileWrite::Write (const Data1D< T > & *data,* const size_t *amnt_to_write*) [inline]

Writes amnt_to_write elements of plain old data type T from data without buffering.

Throws ErrorMsgFile if there was an error writing.

8.28.3.5 template<class T > void RC::FileWrite::Write (const Data1D< T > & *data*) [inline]

Writes all of data's plain old data type T without buffering.

Throws ErrorMsgFile if there was an error writing.

8.28.3.6 void RC::FileWrite::WriteAllStr (const Data1D< RStr > & *lines,* const bool *add_newlines*) [inline]

Writes all lines without buffering, adding a newline after each if add_newlines is true.

Throws ErrorMsgFile if there was a write error.

8.28.3.7 void RC::FileWrite::WriteStr (const char * *str*) [inline]

Writes the null-terminated contents of str without buffering.

Throws ErrorMsgFile if there was a write error.

8.28.3.8 void RC::FileWrite::WriteStr (const RStr & *str*) [inline]

Writes the RStr str without buffering.

Throws ErrorMsgFile if there was a write error.

The documentation for this class was generated from the following file:

- File.h

8.29 RC::HoldRelated< Hold, Provide > Class Template Reference

Stores the value of type Hold while providing access via type Provide.

`#include <RCBits.h>`

Public Member Functions

- HoldRelated (Hold held, Provide give)
  Stores held, but implicitly casts to related value give.
- HoldRelated (Hold held)
  Stores held. Use with Set to set the give value later.
- void Set (Provide new_give)
  Sets the implicitly cast value of type Provide to give.
- operator Provide ()
  Implicitly casts to the give value set.
- operator const Provide () const
  Implicitly casts to the give value set.
- const Provide Get () const
  Explicitly access the value set as give.
- Provide Get ()
  Explicitly access the value set as give.
- const Hold & Held () const
  Access the full held value.
- Hold & Held ()
  Access the full held value.

8.29.1 Detailed Description

template<class Hold, class Provide>
class RC::HoldRelated< Hold, Provide >

Stores the value of type Hold while providing access via type Provide.

See also

RStr::ToLPCWSTR()

The documentation for this class was generated from the following file:

- RCBits.h

8.30 RC::Net::Listener Class Reference

Listens to the specified port for incoming TCP connections.

`#include <Net.h>`

Public Member Functions

- Listener (const RStr &port)

  Begin listening to the specified port for incoming TCP connections.
- ~Listener ()

  If open, this closes the port being listened to.
- SOCKET Raw () const

  Provides the raw socket descriptor being listened to.
- bool Accept (Sock &connection, bool quiet_fail=true)

  Accept an incoming connection, with read/write access through the FileRW connection.

8.30.1 Detailed Description

Listens to the specified port for incoming TCP connections.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 RC::Net::Listener::Listener (const RStr & *port*) [inline]

Begin listening to the specified port for incoming TCP connections.

Use Accept to accept a connection.

8.30.3 Member Function Documentation

8.30.3.1 bool RC::Net::Listener::Accept (Sock & *connection*, bool *quiet_fail* = `true`) [inline]

Accept an incoming connection, with read/write access through the FileRW connection.

This throws an ErrorMsgNet on error if quiet_fail is false.

Returns

False if an error occurred.

The documentation for this class was generated from the following file:

- Net.h

8.31 RC::LoopIndex Class Reference

A size_t like integer class which automatically stays within its range.

`#include <RCBits.h>`

Public Member Functions

- LoopIndex (size_t range)
  Defines the range of of the object. The index is always less than this.
- size_t Range () const
  Returns the set range.
- void SetRange (size_t new_range)
  Sets a new range.
- LoopIndex & operator= (size_t new_index)
  Assigns a new index value, forcing it in the range. Signed indices are handled properly.
- operator size_t () const
  Implicitly returns the size_t index.
- LoopIndex & operator++ ()
  Increment by one, looping within range.
- LoopIndex operator++ (int)
  Postfix increment by one, looping within range.
- LoopIndex & operator-- ()
  Decrement by one, looping within range.
- LoopIndex operator-- (int)
  Postfix decrement by one, looping within range.
- LoopIndex & operator+= (size_t offset)
  Increment by offset, looping within range, and handling negative offsets correctly.
- LoopIndex & operator-= (size_t offset)
  Decrement by offset, looping within range, and handling negative offsets correctly.

8.31.1 Detailed Description

A size_t like integer class which automatically stays within its range.

Use this for circular buffers or anywhere modulo arithmetic is needed.

The documentation for this class was generated from the following file:

- RCBits.h

8.32 RC::Net Class Reference

Provides both client and server sides of blocking TCP connections.

`#include <Net.h>`

Classes

- class Listener

  Listens to the specified port for incoming TCP connections.

Static Public Member Functions

- static void InitializeSockets ()

  Automatically called by Connect and Listener.
- static bool Connect (Sock &connection, const RStr &host, const RStr &port, bool quiet_fail=true)

  Opens a TCP connection to host:port.

8.32.1 Detailed Description

Provides both client and server sides of blocking TCP connections.

8.32.2 Member Function Documentation

8.32.2.1 static bool RC::Net::Connect (Sock & *connection,* const RStr & *host,* const RStr & *port,* bool *quiet_fail* = `true`) `[inline],[static]`

Opens a TCP connection to host:port.

Will throw ErrorMsgNet upon failure to connect if quiet_fail is false.

Returns

True if the connect succeeded.

8.32.2.2 static void RC::Net::InitializeSockets ( ) `[inline],[static]`

Automatically called by Connect and Listener.

The first time this is called, it sets SIGPIPE to ignore so the program doesn't terminate while writing to a closed socket. Call this once manually beforehand if you want to set your own custom handler for SIGPIPE.

The documentation for this class was generated from the following file:

- Net.h

8.33 RC::PluralStr Class Reference

Provides number-based singular/plural string management.

`#include <RStr.h>`

Public Member Functions

- PluralStr (const RStr &singular, const RStr &plural)

  Initialize with the singular and plural version of a string.
- template<class T >
  RStr For (T arg) const

  Provides the singular form if arg is 1, or the plural version otherwise.
- template<class T >
  RStr Count (T arg) const

  For any number arg, provides the number, followed by a space, followed by the correct singular or plural form.

8.33.1 Detailed Description

Provides number-based singular/plural string management.

Example usage: PluralStr cat("cat", "cats"); cout << cat.Count(5) << " plus one " << cat.For(1) << " is 6 " << cat.For(6); Produces: "5 cats plus one cat is 6 cats"

The documentation for this class was generated from the following file:

- RStr.h

8.34 RC::Ptr< T > Class Template Reference

A safe pointer class that throws an RC::ErrorMsgNull if a null dereference is attempted.

`#include <Ptr.h>`

Public Member Functions

- Ptr (T *t_ptr=NULL)

  Default constructor assigning the value of the pointer.
- template<class Tderived >
  Ptr (const APtr< Tderived > &other)

  A constructor which obtains a non-reference-counted copy of an APtr.
- template<class Tderived >
  Ptr (const RevPtr< Tderived > &other)

  A constructor which obtains a non-revokable copy of a RevPtr.
- template<class Tderived >
  Ptr (const Ptr< Tderived > &other)

  Copy constructor.
- void Delete ()

  Deletes the object being pointed to and nulls the pointer.
- T * Raw () const

  Returns a direct reference to the enclosed pointer.
- void Assert () const

  Throws RC::ErrorMsgNull if the pointer is null.
- bool IsSet () const

  True if the pointer is non-NULL.

- bool IsNull () const

  True if the pointer is NULL.
- T & operator* ()

  Dereferences the pointer, or throws an exception if null.
- const T & operator* () const

  Const version of operator()*
- T * operator-> ()

  Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.
- const T * operator-> () const

  Const version of operator->()
- operator T * () const

  Implicitly casts to the enclosed pointer, without null checking.
- template<class Type >
  Ptr< Type > Cast ()

  Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.
- template<class Type >
  const Ptr< Type > Cast () const

  Const version of Cast()
- template<class Type >
  bool CanCast () const

  True if it can dynamically cast to this type.
- template<class Type >
  Type & As ()

  Dynamically casts and dereferences to the type presented as a template parameter, or throws ErrorMsgCast if it fails.
- template<class Type >
  const Type & As () const

  Const version of As()
- T & As ()

  Dereference as the default type.
- const T & As () const

  Const version of operator()*

8.34.1 Detailed Description

template<class T>
class RC::Ptr< T >

A safe pointer class that throws an RC::ErrorMsgNull if a null dereference is attempted.

See also

APtr
RevPtr

8.34.2 Constructor & Destructor Documentation

8.34.2.1 template<class T> RC::Ptr< T >::Ptr (T * *t_ptr* = NULL) [inline]

Default constructor assigning the value of the pointer.

Parameters

t_ptr	The new pointer value.

8.34.2.2 template<class T> template<class Tderived > RC::Ptr< T >::Ptr (const APtr< Tderived > & *other*) [inline]

A constructor which obtains a non-reference-counted copy of an APtr.

Parameters

other	The APtr from which a pointer should be extracted.

8.34.2.3 template<class T> template<class Tderived > RC::Ptr< T >::Ptr (const RevPtr< Tderived > & *other*) [inline]

A constructor which obtains a non-revokable copy of a RevPtr.

Parameters

other	The RevPtr from which a pointer should be extracted.

8.34.2.4 template<class T> template<class Tderived > RC::Ptr< T >::Ptr (const Ptr< Tderived > & *other*) [inline]

Copy constructor.

Parameters

other	The Ptr to copy.

8.34.3 Member Function Documentation

8.34.3.1 template<class T> T& RC::Ptr< T >::As ( ) [inline]

Dereference as the default type.

See also

operator()

8.34.3.2 template<class T> template<class Type > Ptr<Type> RC::Ptr< T >::Cast ( ) [inline]

Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.

The Ptr is NULL if the cast fails.

8.34.3.3 template<class T> bool RC::Ptr< T >::IsNull ( ) const [inline]

True if the pointer is NULL.

Returns

True if the pointer is NULL.

8.34.3.4 template<class T> bool RC::Ptr< T >::IsSet ( ) const [inline]

True if the pointer is non-NULL.

Returns

True if the pointer is non-NULL.

8.34.3.5 template<class T> T& RC::Ptr< T >::operator* ( ) [inline]

Dereferences the pointer, or throws an exception if null.

The exception thrown is RC::ErrorMsgNull.

Returns

A reference to the dereferenced object.

8.34.3.6 template<class T> T* RC::Ptr< T >::operator-> ( ) [inline]

Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.

Returns

The enclosed pointer.

8.34.3.7 template<class T> T* RC::Ptr< T >::Raw ( ) const [inline]

Returns a direct reference to the enclosed pointer.

The reference is read/write, so it can be used as a function parameter which updates the value of this pointer.

Returns

A reference to the enclosed pointer.

The documentation for this class was generated from the following file:

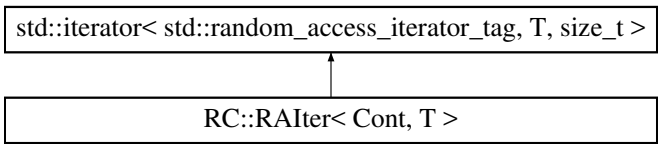
- Ptr.h

8.35 RC::RAIter< Cont, T > Class Template Reference

A bounds-checked iterator for random-access containers that knows when its target was deleted.

`#include <Iter.h>`

Inheritance diagram for RC::RAIter< Cont, T >:



Public Member Functions

- RAIter ()
  Default constructor to nothing, by default invalid for use.
- RAIter (RevPtr< Cont > container, size_t pos)
  Creates an iterator to a container starting at index pos.
- bool IsValid () const
  True if the iterator still points to an existing container and remains in bounds.
- size_t GetIndex () const
  Return the index of the container which this iterator corresponds to.
- RevPtr< Cont > Raw ()
  Directly access the RevPtr to the container.
- T & operator* ()
  Dereference the iterator, returning a reference to the corresponding element.
- const T & operator* () const
  Const version of operator.*
- const T * operator-> () const
  Provides access to members of the element this iterator points to.
- T & operator[ ] (size_t x)
  Provides access to the element at offset x from the current index.
- const T & operator[ ] (size_t x) const
  Const version of operator[].
- RAIter< Cont, T > & operator++ ()
  Point to the next higher numbered indexed element.
- RAIter< Cont, T > & operator-- ()
  Point to the next lower numbered indexed element.
- RAIter< Cont, T > & operator+= (size_t x)
  Increase the index by x.
- RAIter< Cont, T > & operator-= (size_t x)
  Decrease the index by x.
- RAIter< Cont, T > operator++ (int)
  Support for iter++, but for efficiency, use ++iter.
- RAIter< Cont, T > operator-- (int)
  Support for iter–, but for efficiency, use –iter.
- RAIter< Cont, T > operator+ (size_t x) const
  Returns a new iterator with an offset incremented by x.

- RAIter< Cont, T > operator- (size_t x) const

  Returns a new iterator with an offset decremented by x.
- size_t operator- (const RAIter< Cont, T > &first) const

  Returns the distance between two iterators.
- bool operator== (const RAIter< Cont, T > &other) const

  True if the two iterators have the same index.
- bool operator< (const RAIter< Cont, T > &other) const

  True if this iterator is less.
- template<class AnyValidType >
  bool operator!= (const AnyValidType &other) const

  True if not equal.
- template<class AnyValidType >
  bool operator> (const AnyValidType &other) const

  True if other is less than this object.
- template<class AnyValidType >
  bool operator<= (const AnyValidType &other) const

  True if other is not less than this object.
- template<class AnyValidType >
  bool operator>= (const AnyValidType &other) const

  True if this object is not less than other.

8.35.1 Detailed Description

template<class Cont, class T>
class RC::RAIter< Cont, T >

A bounds-checked iterator for random-access containers that knows when its target was deleted.

See Data1D.h for a usage example.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 template<class Cont, class T> RC::RAIter< Cont, T >::RAIter (RevPtr< Cont > *container*, size_t *pos*) [inline]

Creates an iterator to a container starting at index pos.

When the iterator is dereferenced, it corresponds to (∗container)[pos]. Note that the RevPtr for container should be set to AutoRevoke when the container is deleted.

Parameters

container	A RevPtr to the container.
pos	The starting index for this iterator.

The documentation for this class was generated from the following file:

- Iter.h

8.36 RC::RevPtr< T > Class Template Reference

A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.

`#include <RevPtr.h>`

Public Member Functions

- RevPtr (T *t_ptr=NULL)

  Default constructor assigning the value of the pointer.
- RevPtr (const RevPtr< T > &other)

  Copy constructor.
- RevPtr (const Ptr< T > &other)

  A conversion constructor which creates a new RevPtr from a Ptr of the same or a derived type.
- RevPtr (const APtr< T > &other)

  A conversion constructor which creates a new RevPtr from an APtr of the same or a derived type.
- RevPtr & operator= (const RevPtr< T > &other)

  An assignment operator which joins this RevPtr to the other RevPtr.
- ~RevPtr ()

  Destructor which revokes the pointer only if AutoRevoke() was used on this object.
- void Delete ()

  Manually delete the object pointed to by this RevPtr, and revoke the pointer for all shared RevPtr's.
- void Revoke ()

  Manually revoke the pointer, setting it equal to NULL for all shared RevPtr objects.
- void AutoRevoke (bool new_auto_revoke=true)

  Set this RevPtr to revoke the pointer upon deletion or leaving scope.
- T * Raw () const

  Returns a direct reference to the enclosed pointer.
- void Assert () const

  Throws RC::ErrorMsgNull if the pointer is null.
- bool IsSet () const

  True if the pointer is non-NULL.
- bool IsNull () const

  True if the pointer is NULL.
- T & operator* ()

  Dereferences the pointer, or throws an exception if null.
- const T & operator* () const

  Const version of operator()*
- T * operator-> ()

  Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.
- const T * operator-> () const

  Const version of operator->()
- operator T * () const

  Implicitly casts to the enclosed pointer, without null checking.
- template<class Type >
  Ptr< Type > Cast ()

  Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.
- template<class Type >
  const Ptr< Type > Cast () const

  Const version of Cast()

- template<class Type >
  bool CanCast () const

  True if it can dynamically cast to this type.
- template<class Type >
  Type & As ()

  Dynamically casts and dereferences to the type presented as a template parameter, or throws ErrorMsgCast if it fails.
- template<class Type >
  const Type & As () const

  Const version of As()
- T & As ()

  Dereference as the default type.
- const T & As () const

  Const version of operator()*

8.36.1 Detailed Description

template<class T>
class RC::RevPtr< T >

A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.

This pointer class does not automatically delete the object pointed to, so that it can be used for self-referential purposes. For automatic deletion, use an APtr, and create a RevPtr family that refers to the Ptr of the same scope. Like Ptr and APtr, dereferencing a null RevPtr throws ErrorMsgNull. Note: Revocation is thread safe, but a previously extracted raw ptr could remain in use after revocation. Be mindful of this in threading architecture design.

See also

Ptr
APtr

8.36.2 Constructor & Destructor Documentation

8.36.2.1 template<class T> RC::RevPtr< T >::RevPtr (T * *t_ptr* = NULL) [inline]

Default constructor assigning the value of the pointer.

Parameters

t_ptr	The new pointer value.

8.36.2.2 template<class T> RC::RevPtr< T >::RevPtr (const RevPtr< T > & *other*) [inline]

Copy constructor.

Parameters

other	The RevPtr to copy.

8.36.2.3 template<class T> RC::RevPtr< T >::RevPtr (const Ptr< T > & *other*) [inline]

A conversion constructor which creates a new RevPtr from a Ptr of the same or a derived type.

Parameters

other	The Ptr from which the pointer should be used to make this RevPtr

8.36.2.4 template<class T> RC::RevPtr< T >::RevPtr (const APtr< T > & *other*) [inline]

A conversion constructor which creates a new RevPtr from an APtr of the same or a derived type.

Parameters

other	The APtr from which the pointer should be used to make this RevPtr

8.36.3 Member Function Documentation

8.36.3.1 template<class T> T& RC::RevPtr< T >::As ( ) [inline]

Dereference as the default type.

See also

operator()

8.36.3.2 template<class T> void RC::RevPtr< T >::AutoRevoke (bool *new_auto_revoke* = true) [inline]

Set this RevPtr to revoke the pointer upon deletion or leaving scope.

Parameters

new_auto_revoke	True if this RevPtr should automatically revoke. The recommended way to use this feature is to place the primary RevPtr in scope with the object to which it points so that they are deleted together, and then call AutoRevoke() on that RevPtr. Then all other RevPtr's which have been assigned or copy constructed from the primary RevPtr will be automatically revoked when it is destructed, and they will all return true for IsNull or throw exceptions if dereferencing is attempted.

8.36.3.3 template<class T> template<class Type > Ptr<Type> RC::RevPtr< T >::Cast ( ) [inline]

Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.

The Ptr is NULL if the cast fails.

8.36.3.4 template<class T> void RC::RevPtr< T >::Delete ( ) [inline]

Manually delete the object pointed to by this RevPtr, and revoke the pointer for all shared RevPtr's.

Note: While this NULLs all the linked RevPtr's, it cannot affect raw pointers which have already been extracted or are in use when this is called. Be mindful of this in threading architecture design.

8.36.3.5 template<class T> bool RC::RevPtr< T >::IsNull ( ) const [inline]

True if the pointer is NULL.

Returns

True if the pointer is NULL.

8.36.3.6 template<class T> bool RC::RevPtr< T >::IsSet ( ) const [inline]

True if the pointer is non-NULL.

Returns

True if the pointer is non-NULL.

8.36.3.7 template<class T> T& RC::RevPtr< T >::operator* ( ) [inline]

Dereferences the pointer, or throws an exception if null.

The exception thrown is RC::ErrorMsgNull.

Returns

A reference to the dereferenced object.

8.36.3.8 template<class T> T* RC::RevPtr< T >::operator-> ( ) [inline]

Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.

Returns

The enclosed pointer.

8.36.3.9 template<class T> RevPtr& RC::RevPtr< T >::operator= (const RevPtr< T > & *other*) [inline]

An assignment operator which joins this RevPtr to the other RevPtr.

Parameters

other	The source RevPtr to assign here.

8.36.3.10 template<class T> T* RC::RevPtr< T >::Raw ( ) const [inline]

Returns a direct reference to the enclosed pointer.

The reference is read/write, so it can be used as a function parameter which updates the value of this pointer.

Returns

A reference to the enclosed pointer.

The documentation for this class was generated from the following file:

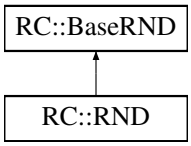
- RevPtr.h

8.37 RC::RND Class Reference

A Mersenne Twister random number generator class with integer, array, or time-based seeding.

`#include <RND.h>`

Inheritance diagram for RC::RND:



Public Member Functions

- void Seed (u32 seed)
  Reseed the random number generator with the value seed.
- void Seed (const Data1D< u32 > &init_key)
  Reseed the generator with up to the first 624 values of init_key.
- RND (u32 seed)
  Construct the generator with the initial seed given.
- RND (const Data1D< u32 > &init_key)
  Construct the generator with the initial seed being the first 624 values of init_key.
- RND ()
  Default constructor, which seeds using the system time and 64 bits of environmental entropy.
- void Seed_urandom ()
  Seed the genederator by reading 624 u32's from /dev/urandom.
- virtual u32 Get_u32 ()
  Provides random u32 values.

Static Public Attributes

- static const u32 CONST_SEED =5489

  The reference implementation's recommended seed.

Additional Inherited Members

8.37.1 Detailed Description

A Mersenne Twister random number generator class with integer, array, or time-based seeding.

The output of this class is compliant with the mt19937ar reference implementation by Nishimura & Matsumoto.

The documentation for this class was generated from the following file:

- RND.h

8.38 RC::RStr Class Reference

A bounds-safe string class which provides an identical interface to std::string plus many convenience functions for string manipulation.

```
#include <RStr.h>
```

Wrapper methods for std::string

- RStr ()

  Default constructor, initializes to blank string.
- RStr (const RStr &other)

  Copy constructor.
- RStr (const std::string &str)

  Initializes to str.
- RStr (const RStr &s, size_t pos, size_t n=npos)

  Creates a new RStr from a segment of s starting at index pos with length n or until the end.
- RStr (const char *s, size_t n)

  Creates a new RStr from n characters at s, or until null-termination.
- RStr (const char *s)

  Creates a new RStr from s until null-termination, or empty string if s is NULL.
- RStr (size_t n, char c)

  Initializes to n copies of character c.
- template<class InputIterator >
  RStr (InputIterator begin, InputIterator end)

  Initializes the string from the values in the iterator range.
- RStr (const std::initializer_list< char > characters)

  Initializes the string with the list of characters.
- RStr (RStr &&other) noexcept

  Moves other to this RStr.
- RStr & operator= (const RStr &s)

Copies the contents of s to this RStr.

- RStr & operator= (const char *s)

  Copies the null-terminated characters at s to this RStr, or empties if s is NULL.
- RStr & operator= (char c)

  Sets the RStr to char c.
- RStr & operator= (std::initializer_list< char > characters)

  Sets the contents of the RStr to the initializer list.
- RStr & operator= (RStr &&other)

  Moves the contests of other to this RStr.
- RStrIter begin ()

  Returns a bounds-safe iterator to the first element.
- RStrIter end ()

  Returns a bounds-safe iterator which points past the last element.
- size_t size () const

  Returns the length of the string.
- size_t length () const

  Returns the length of the string.
- size_t max_size () const

  Returns the maximum size of the string.
- void resize (size_t n, char c)

  Resizes the string, filling with character c if enlarging.
- void resize (size_t n)

  Resizes the string, filling with null characters if enlarging.
- size_t capacity () const

  Returns the allocated storage space.
- void reserve (size_t capsize=0)

  Request a storage capacity increase to capsize.
- void clear ()

  Make the string empty.
- bool empty () const

  True of the string is empty.
- char & operator[ ] (size_t pos)

  Returns the char at index pos, or throws ErrorMsgBounds if out of bounds.
- const char & operator[ ] (size_t pos) const

  Const version of operator[].
- char & at (size_t pos)

  Identical to operator[].
- const char & at (size_t pos) const

  Const version of at.
- RStr & operator+= (const RStr &s)

  Appends s to this RStr.
- RStr & operator+= (const char *s)

  Appends s to this RStr if s is non-NULL.
- RStr & operator+= (char c)

  Appends the char c to this RStr.
- RStr & append (const RStr &s)

  Appends s to this RStr.
- RStr & append (const RStr &s, size_t pos, size_t n)

  Appends n characters in s starting at pos to this RStr, or fewer characters if s is too short.
- RStr & append (const char *s, size_t n)

  Appends n characters starting at s to this RStr, or fewer if null-termination is reached in s.

- RStr & append (const char *s)

  Append s to this RStr if s is non-NULL.
- RStr & append (size_t n, char c)

  Appends n copies of char c.
- template<class InputIterator >
  RStr & append (InputIterator first, InputIterator last)

  Appends from first to one before last, or until a null character.
- RStr & append (std::initializer_list< char > characters)

  Append the initializer list of characters.
- void push_back (char c)

  Appends character c.
- RStr & assign (const RStr &s)

  Copies s to this RStr.
- RStr & assign (const RStr &s, size_t pos, size_t n)

  Sets this RStr to n characters in s at index pos, or until the end.
- RStr & assign (const char *s, size_t n)

  Sets this RStr to n characters starting at s or until null-termination.
- RStr & assign (const char *s)

  Sets this RStr to the characters at s until null-termination.
- RStr & assign (size_t n, char c)

  Sets this RStr to n copies of character c.
- template<class InputIterator >
  RStr & assign (InputIterator first, InputIterator last)

  Sets the RStr to characters from first to one before last, or until a null character.
- RStr & insert (size_t pos_this, const RStr &s)

  Inserts s starting at index pos_this of this RStr, or throws ErrorMsgBounds.
- RStr & insert (size_t pos_this, const RStr &s, size_t pos_s, size_t n)

  Inserts at index pos_this, n characters from s starting at pos_s.
- RStr & insert (size_t pos_this, const char *s, size_t n)

  Inserts at index pos_this, n characters from s or until NULL-termination.
- RStr & insert (size_t pos_this, const char *s)

  Inserts s at index pos_this until null-termination, or nothing if s is NULL.
- RStr & insert (size_t pos_this, size_t n, char c)

  Inserts n copies of char c at pos_this.
- RStrIter insert (RStrIter p, char c)

  Inserts char c at iterator position p.
- void insert (RStrIter p, size_t n, char c)

  Inserts n copies of char c at iterator position p.
- template<class InputIterator >
  void insert (RStrIter p, InputIterator first, InputIterator last)

  At iterator position p, inserts characters from first through one before last or until null-termination.
- RStr & insert (const RStrIter p, std::initializer_list< char > characters)

  At iterator position p, inserts list of characters.
- RStr & erase (size_t pos=0, size_t n=npos)

  Erases n characters at index pos, or until the end.
- RStrIter erase (const RStrIter pos)

  Erases one character at index pos.
- RStrIter erase (const RStrIter first, const RStrIter last)

  Erases from index first through one before index last.
- RStr & replace (size_t pos_this, size_t n_this, const RStr &s)

  Replace n_this characters at pos_this with all of s.

- RStr & replace (RStrIter first, RStrIter last, const RStr &s)

  Replace from first to one before last with all of s.
- RStr & replace (size_t pos_this, size_t n_this, const RStr &s, size_t pos_s, size_t n_s)

  Replace n_this characters at pos_this with n_s characters from s starting at pos_s.
- RStr & replace (size_t pos_this, size_t n_this, const char *s, size_t n_s)

  Replace n_this characters at pos_this with n_s characters from s, or until s's null-termination.
- RStr & replace (RStrIter first, RStrIter last, const char *s, size_t n_s)

  Replace from first through one before last with n_s characters from s, or until s's null-termination.
- RStr & replace (size_t pos_this, size_t n_this, const char *s)

  Replace n_this characters at pos_this with all of s, or nothing if s is NULL.
- RStr & replace (RStrIter first, RStrIter last, const char *s)

  Replace from first to one before last with all of s, or nothing if s is NULL.
- RStr & replace (size_t pos_this, size_t n_this, size_t n_c, char c)

  Replace n_this characters at pos_this with n_c copies of c.
- RStr & replace (RStrIter first, RStrIter last, size_t n_c, char c)

  Replace from first through one before last with n_c copies of c.
- template<class InputIterator >
  RStr & replace (RStrIter first, RStrIter last, InputIterator in_first, InputIterator in_last)

  Replace from first through one before lsat with in1 through one before in2, or until null-termination.
- void swap (RStr &s)

  Swap contents with s.
- const char * c_str () const

  Provides a null-terminated C style string corresponding to RStr.
- const char * data () const

  Identical to c_str.
- std::allocator< char > get_allocator () const

  Get the allocator used for string storage.
- size_t copy (char *s, size_t n, size_t pos=0) const

  Copies n characters to s, starting from pos_this.
- size_t find (const RStr &s, size_t pos=0) const

  Returns the index at pos or later which matches string s, or npos if no match.
- size_t find (const char *s, size_t pos, size_t n) const

  Returns the index at pos or later which matches n characters of s, or npos if no match.
- size_t find (const char *s, size_t pos=0) const

  Returns the index at pos or later which matches s until null-termination, or npos if no match.
- size_t find (char c, size_t pos=0) const

  Returns the index at pos or later which matches character c.
- size_t rfind (const RStr &s, size_t pos=npos) const

  Returns the index at pos or before which starts a match of string s, or npos if no match.
- size_t rfind (const char *s, size_t pos, size_t n) const

  Returns the index at pos or before which matches n characters of s, or npos if no match.
- size_t rfind (const char *s, size_t pos=npos) const

  Returns the index at pos or before which matches s until null-termination, or npos if no match.
- size_t rfind (char c, size_t pos=npos) const

  Returns the index at pos or before which matches character c.
- size_t find_first_of (const RStr &s, size_t pos=0) const

  Returns the first index at pos or later which matches a character in s, or npos if none match.
- size_t find_first_of (const char *s, size_t pos, size_t n) const

  Returns the first index at pos or later which matches one of the first n characters in s, or npos if none match.
- size_t find_first_of (const char *s, size_t pos=0) const

  Returns the first index at pos or later which matches a character in s, or npos if none match.

- size_t find_first_of (char c, size_t pos=0) const

  Returns the index at pos or later which matches character c.
- size_t find_last_of (const RStr &s, size_t pos=npos) const

  Returns the highest index at pos or before which matches a character in s, or npos if none match.
- size_t find_last_of (const char *s, size_t pos, size_t n) const

  Returns the highest index at pos or before which matches the first n characters in s, or npos if none match.
- size_t find_last_of (const char *s, size_t pos=npos) const

  Returns the highest index at pos or before which matches a character in s, or npos if none match.
- size_t find_last_of (char c, size_t pos=npos) const

  Returns the highest index at pos or before which matches character c, or npos if none match.
- size_t find_first_not_of (const RStr &s, size_t pos=0) const

  Returns the first index at pos or later which does not match a character in s, or npos if all match.
- size_t find_first_not_of (const char *s, size_t pos, size_t n) const

  Returns the first index at pos or later which does not match the first n characters in s, or npos if all match.
- size_t find_first_not_of (const char *s, size_t pos=0) const

  Returns the first index at pos or later which does not match a character in s, or npos if all match.
- size_t find_first_not_of (char c, size_t pos=0) const

  Returns the first index at pos or later which does not match character c, or npos if all match.
- size_t find_last_not_of (const RStr &s, size_t pos=npos) const

  Returns the highest index at pos or before which does not match a character in s, or npos if all match.
- size_t find_last_not_of (const char *s, size_t pos, size_t n) const

  Returns the highest index at pos or before which does not match the first n characters in s, or npos if all match.
- size_t find_last_not_of (const char *s, size_t pos=npos) const

  Returns the highest index at pos or before which does not match a character in s, or npos if all match.
- size_t find_last_not_of (char c, size_t pos=npos) const

  Returns the highest index at pos or before which does not match character c, or npos if all match.
- RStr substr (size_t pos=0, size_t n=npos) const

  Creates a substring from n characters starting at position pos, or until the end of the string.
- int compare (const RStr &s) const

  Returns negative, 0, or positive if this string is lesser, equal, or greater than s.
- int compare (const char *s) const

  Returns negative, 0, or positive if this string is lesser, equal, or greater than s.
- int compare (size_t pos_this, size_t n_this, const RStr &s) const

  Returns negative, 0, or positive if the n_this characters at pos_this are lesser, equal, or greater than s.
- int compare (size_t pos_this, size_t n_this, const char *s) const

  Returns negative, 0, or positive if the n_this characters at pos_this are lesser, equal, or greater than s.
- int compare (size_t pos_this, size_t n_this, const RStr &s, size_t pos_s, size_t n_s) const

  Returns negative, 0, or positive if the n_this characters at pos_this are lesser, equal, or greater than the n_s characters at pos_s in s.
- int compare (size_t pos_this, size_t n_this, const char *s, size_t n_s) const

  Returns negative, 0, or positive if the n_this characters at pos_this are lesser, equal, or greater than the n_s characters in s.

Convert to and from other supported types

- RStr (char x)

  The default constructor for a char, treating it as a character.
- RStr (char x, RStr_IntStyle style, i32 precision=-1)

  Formats x as a string in the given style, and with at least precision 0-padded digits.
- RStr (u8 x, RStr_IntStyle style=DEC, i32 precision=-1)

Formats x as a string in the given style, and with at least \ precision 0-padded digits.

- RStr (i8 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (u16 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (i16 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (u32 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (i32 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (u64 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (i64 x, RStr_IntStyle style=DEC, i32 precision=-1)

  Formats x as a string in the given style, and with at least \ precision 0-padded digits.
- RStr (f32 x, RStr_FloatStyle style=AUTO, u32 precision=std::numeric_limits< f32 >::digits10)

  Formats x as a string in the given style, and with it rounded to precision digits.
- RStr (f64 x, RStr_FloatStyle style=AUTO, u32 precision=std::numeric_limits< f64 >::digits10)

  Formats x as a string in the given style, and with it rounded to precision digits.
- RStr (f80 x, RStr_FloatStyle style=AUTO, u32 precision=std::numeric_limits< f80 >::digits10)

  Formats x as a string in the given style, and with it rounded to precision digits.
- RStr (bool b)

  "true" if b is true, or "false" if b is false.
- RStr (bool pad0s, void *ptr, bool use0x=true)

  Constructor for displaying a pointer, with 0's prepended if pad0s is true, and 0x prepended if use0x is true.
- template<class T >
  RStr (const Data1D< T > &arr)

  Initialize the string with the bytes stored in Data1D treated as raw character data.
- RStr (const std::wstring &wstr)

  Convert a std::wstring to RStr, discarding the high bits.
- RStr (const QString &qstr)

  Convert a QString to RStr, discarding the high bits.
- f32 Get_f32 () const

  Convert the beggining characters of this string to a float.
- f64 Get_f64 () const

  Convert the beggining characters of this string to a float.
- f80 Get_f80 () const

  Convert the beggining characters of this string to a float.
- u32 Get_u32 (int base=0) const

  Convert the beggining characters of this string to this integer type.
- u64 Get_u64 (int base=0) const

  Convert the beggining characters of this string to this integer type.
- i32 Get_i32 (int base=0) const

  Convert the beggining characters of this string to this integer type.
- i64 Get_i64 (int base=0) const

  Convert the beggining characters of this string to this integer type.
- u32 Get_hex32 () const

  Convert the beggining characters of this string as a hexadecimal to this integer type.
- u64 Get_hex64 () const

  Convert the beggining characters of this string as a hexadecimal to this integer type.
- bool Get_bool () const

Returns true if case-insensitive "true", "T", or non-zero.

- void Get (f32 &x) const

  Overloaded function to extract type f32 from this class.
- void Get (f64 &x) const

  Overloaded function to extract type f64 from this class.
- void Get (f80 &x) const

  Overloaded function to extract type f80 from this class.
- void Get (u32 &x) const

  Overloaded function to extract type u32 from this class.
- void Get (u64 &x) const

  Overloaded function to extract type u64 from this class.
- void Get (i32 &x) const

  Overloaded function to extract type i32 from this class.
- void Get (i64 &x) const

  Overloaded function to extract type i64 from this class.
- void Get (bool &x) const

  Overloaded function to extract type bool from this class.
- bool Is_f32 (bool strict=false) const

  True if the start of the string can be converted to an f32.
- bool Is_f64 (bool strict=false) const

  True if the start of the string can be converted to an f64.
- bool Is_f80 (bool strict=false) const

  True if the start of the string can be converted to an f80.
- bool Is_u32 (int base=0, bool strict=false) const

  True if the start of the string can be converted to an u32.
- bool Is_u64 (int base=0, bool strict=false) const

  True if the start of the can be converted to an u64.
- bool Is_i32 (int base=0, bool strict=false) const

  True if the start of the string can be converted to an i32.
- bool Is_i64 (int base=0, bool strict=false) const

  True if the start of the string can be converted to an i64.
- bool Is_hex32 (bool strict=false) const

  True if the start of the string can be converted as hexadecimal to a u32.
- bool Is_hex64 (bool strict=false) const

  True if the start of the string can be converted as hexadecimal to a u64.
- bool Is_bool () const

  True if the string is a "0", "1", or case insensitive "true", "false", "T", or "F".
- QString ToQString () const

  Convert the RStr to a QString.
- std::wstring wstring ()

  Convert the RStr to a std::wstring.
- const char * ToLPCSTR () const

  For Win32, provides an LPCSTR to the string.
- HoldRelated< std::wstring, const wchar_t * > ToLPCWSTR ()

  For Win32, provides an LPCWSTR to the string.
- const char * ToLPCTSTR ()

  For Win32, provides an LPCTSTR to the string.
- std::string & Raw ()

  Provides raw access to the std::string this RStr wraps.
- const std::string & Raw () const

  Provides const raw access to the std::string this RStr wraps.
- Data1D< char > ToData () const

  Returns a Data1D<char> corresponding to the character data in the string.

Bounds-checking

- bool Check (size_t pos) const

  True if the index pos is in bounds.
- bool CheckPlus (size_t pos) const

  True if the index pos is in bounds, permitting the null.
- void Assert (size_t pos) const

  Throws ErrorMsgBounds if the index pos is out of bounds.
- void AssertPlus (size_t pos) const

  Throws ErrorMsgBounds if the index pos is out of bounds, permitting the null.
- void Assert (const char *ptr) const

  Throws ErrorMsgNull if ptr is null.

Text alignment and arrangement functions

- RStr & PadLeft (const size_t pad_to, const char pad_with=' ')

  If the length is less than pad_to, add pad_width to the left until it reaches that size.
- RStr & PadRight (const size_t pad_to, const char pad_with=' ')

  If the length is less than pad_to, add pad_width to the right until it reaches that size.
- RStr & PadCenter (const size_t pad_to, const char pad_with=' ')

  If the length is less than pad_to, add pad_width evenly to the right and left until it reaches that size.
- Data1D< RStr > Wrap (const size_t width)

  Wraps each line in this RStr to be no longer than width.
- Data1D< RStr > WordWrap (const size_t width)

  Wraps each line in this RStr to be no longer than width, but tries to keep whole words together.

Regular expression support

These use C++11 regex by default, but can fall back on boost regex if C++11 is missing and RC_HAVE_BOOST is defined in RCconfig.h.

- bool Match (const RC_REGEX_NS::regex &reg)

  True if the regular expression reg matches this string.
- bool Match (const RStr &regstr)

  True if the regular expression regstr matches this string.
- bool Match (const RC_REGEX_NS::regex &reg, Data1D< RStr > &matches)

  True if the regular expression reg matches this string, and returns the matches in matches.
- bool Match (const RStr &regstr, Data1D< RStr > &matches)

  True if the regular expression regstr matches this string, and returns the matches in matches.
- void Subst (const RC_REGEX_NS::regex &reg, const RStr &subst)

  Substitutes regular expression reg with subst in this string.
- void Subst (const RStr &regstr, const RStr &subst)

  Substitutes regular expression regstr with subst in this string.

Text manipulation routines

- bool Contains (const RStr &s, size_t pos=0) const

  Returns true if pos or later which matches string s.
- size_t After (const RStr &s, size_t pos=0) const

  Returns the index after the first instance of string s at pos or later, or npos if no match.
- RStr & Chomp (const RStr &chomp_chars="\r\n")

  Remove all trailing newline characters, or provided char set.
- RStr & Trim (const RStr &trim_chars=" \t\r\n")

  Remove all leading and trailing whitespace, or provided char set.
- RStr & Truncate (const size_t max_size)

  Make sure the string is no longer than max_size.
- RStr & ToLower ()

  Make this RStr lowercase.
- RStr & ToUpper ()

  Make this RStr uppercase.
- Data1D< RStr > Split (char c) const

  Return an array of strings split at each instance of c.
- Data1D< RStr > SplitFirst (const RStr &dividers) const

  Return an array of 2 strings split at the first character from dividers.
- Data1D< RStr > SplitLast (const RStr &dividers) const

  Return an array of 2 strings split at the last character from dividers.
- Data1D< RStr > SplitAny (const RStr &dividers) const

  Return an array of strings split by any characters in dividers.
- Data1D< RStr > SplitWords () const

  Return an array of one or more whitespace separated words.
- template<class T >
  static RStr Join (const Data1D< T > &str_arr, const RStr &spacer="")

  Takes a Data1D<T> array and joins them as one string with spacer between each element.

Metrics

- size_t Distance (const RStr &other) const

  Compute the Levenshtein distance to other.
- size_t Length8 () const

  Determine the length of the contents treated as a UTF-8 string.

Interacting with standard formats

- Data1D< RStr > SplitCSV () const

  Return comma-separated values with whitespace trimmed.
- RStr ISOtoUTF8 () const

  Treat this string as ISO-8859-1 and return a UTF8 string.
- Data1D< u32 > UTF8toUTF32 () const

  Treat the contents as a UTF-8 string, and return corresponding UTF-32 data.
- template<class T >
  static RStr MakeCSV (const Data1D< T > &arr)

  Makes a comma-separated values string out of a Data1D<T> arr.
- template<class T >
  static RStr MakeCSV (const Data2D< T > &arr)

Makes a comma-separated values string out of a Data2D<T> arr.

- template<class T >
  static RStr MakeCSV (const Data3D< T > &arr)

  Makes a comma-separated values string out of a Data3D<T> arr.
- static RStr ToBase64 (const Data1D< u8 > &input, size_t wrap=0)

  Encodes input to standard base64 in compliance with RFC 4648.
- static RStr ToBase64URL (const Data1D< u8 > &input, size_t wrap=0)

  Encodes input to base64url in compliance with RFC 4648, safe for url and filename use.
- static Data1D< u8 > FromBase64 (const RStr &input)

  Decodes input from base64 in compliance with RFC 4648, standard or URL encoding.
- static Data1D< u8 > FromHexStr (const RStr &hexstr)

  Converts hexstr to a byte array, like "a4c208" -> { 0xa4, 0xc2, 0x08 }.
- template<class T >
  static RStr ToHexStr (const Data1D< T > &rawdata)

  Converts rawdata to a hex string, like { 0xa4, 0xc2, 0x08 } -> "a4c208".
- static Data1D< RStr > Args (int argc, char *argv[])

  Converts standard arguments to "int main" into a Data1D<RStr>.
- static RStr Errno (int err_num)

  Returns a thread-safe error string corresponding to err_num.
- static RStr Errno ()

  Returns a thread-safe error string corresponding to errno.

Stream interfacing

- bool GetLine (std::istream &in=std::cin, char delim='\n')

  Set the string equal to the next line from istream in, up to character delim.
- std::istream & operator>> (std::istream &in, RStr &rstr)

  Input text from a std::istream (e.g., std::cin) into an RStr.
- std::ostream & operator<< (std::ostream &out, const RStr &rstr)

  Output text from an RStr to a std::ostream (e.g., std::cout).

Constants

- static const size_t npos = -1

  The largest possible value of size_t.
- static const RStr Whitespace ()

  Provides all the standard whitespace characters.
- static const RStr NewLine ()

  Provides the OS-specific newline string.

8.38.1 Detailed Description

A bounds-safe string class which provides an identical interface to std::string plus many convenience functions for string manipulation.

This class checks boundaries on access and sanitizes inputs whenever possible, throwing ErrorMsgBounds or ErrorMsgNull when a usage problem arises. Functions are provided for converting to and from numerical types with formatting control, as well as high-level functions for splitting, joining, regular expressions, word-wrapping and alignment, and working with base64, UTF-8, and CSV.

Usage example:

```
u64 val = 0x12ab34cd5;

// 5011360981 in hex is 0x000000012ab34cd5, or 12ab34cd5
cout << val << " in hex is " << RStr(val, HEX0x)
     << ", or " << RStr(val, HEX, 1) << endl;

// Its reciprocal is 1.9955e-10
RStr message = "Its reciprocal is " + RStr(1.0/val, SCI, 5) + "\n";
cout << message;

// Pi dollars would be $3.14
cout << "Pi dollars would be $" << RStr(3.14159, FIXED, 2) << endl;

// foo, bar, baz
cout << RStr::Join(RStr("foo:bar:baz").Split(':'), ", ") << endl;
```

8.38.2 Constructor & Destructor Documentation

8.38.2.1 RC::RStr::RStr (const RStr & *s*, size_t *pos*, size_t *n* = npos) [inline]

Creates a new RStr from a segment of s starting at index pos with length n or until the end.

Throws RC::ErrorMsgBounds if pos is out of bounds.

8.38.2.2 RC::RStr::RStr (const char * *s*, size_t *n*) [inline]

Creates a new RStr from n characters at s, or until null-termination.

Creates an empty string if s is NULL.

8.38.2.3 RC::RStr::RStr (char *x*) [inline]

The default constructor for a char, treating it as a character.

To override, cast to another type or add an RStr_IntStyle parameter.

8.38.2.4 RC::RStr::RStr (char *x*, RStr_IntStyle *style*, i32 *precision* = -1) [inline]

Formats x as a string in the given style, and with at least precision 0-padded digits.

Note, the default style for a char is as a character.

8.38.2.5 RC::RStr::RStr (f32 *x*, RStr_FloatStyle *style* = AUTO, u32 *precision* = std::numeric_limits< f32 >::digits10) [inline]

Formats x as a string in the given style, and with it rounded to precision digits.

For SCI precision is the significant digits to show, for AUTO it is the most signicant digits to show before removing trailing zeroes, and for FIXED, precision is digits after the decimal. AUTO has no exponent for numbers in the range (1e-5, 10^Precision).

8.38.2.6 RC::RStr::RStr (f64 *x*, RStr_FloatStyle *style* = AUTO, u32 *precision* = std::numeric_limits< f64 >::digits10) [inline]

Formats x as a string in the given style, and with it rounded to precision digits.

For SCI precision is the significant digits to show, for AUTO it is the most signicant digits to show before removing trailing zeroes, and for FIXED, precision is digits after the decimal. AUTO has no exponent for numbers in the range (1e-5, 10^Precision).

8.38.2.7 RC::RStr::RStr (f80 *x*, RStr_FloatStyle *style* = AUTO, u32 *precision* = std::numeric_limits< f80 >::digits10) [inline]

Formats x as a string in the given style, and with it rounded to precision digits.

For SCI precision is the significant digits to show, for AUTO it is the most signicant digits to show before removing trailing zeroes, and for FIXED, precision is digits after the decimal. AUTO has no exponent for numbers in the range (1e-5, 10^Precision).

8.38.3 Member Function Documentation

8.38.3.1 RStr& RC::RStr::append (const char * *s*, size_t *n*) [inline]

Appends n characters starting at s to this RStr, or fewer if null-termination is reached in s.

Appends nothing if s is NULL.

8.38.3.2 static Data1D<RStr> RC::RStr::Args (int *argc*, char * *argv[]*) [inline], [static]

Converts standard arguments to "int main" into a Data1D<RStr>.

Index 0 contains the program name, corresponding to argv[0], and index 1 on up are the parameters. Thus size()-1 of the returned value is the number of command-line parameters received.

8.38.3.3 RStr& RC::RStr::assign (const char * *s*, size_t *n*) [inline]

Sets this RStr to n characters starting at s or until null-termination.

The string is cleared if s is NULL.

8.38.3.4 RStr& RC::RStr::assign (const char * *s*) [inline]

Sets this RStr to the characters at s until null-termination.

The string is cleared if s is NULL.

8.38.3.5 RStrIter RC::RStr::begin ( ) [inline]

Returns a bounds-safe iterator to the first element.

The iterator throws ErrorMsgNull if it is accessed after this RStr is destructed. The element pointed to by this respects the offset with SetOffset.

8.38.3.6 size_t RC::RStr::copy (char * *s*, size_t *n*, size_t *pos* = 0) const [inline]

Copies n characters to s, starting from pos_this.

Warning: If n is greater than the length of s, this routine can write past the end of s. Consider instead obtaining a managed Data1D<char> via ToData(), using Data1D::CopyAt, and accessing the char* with Data1D::Raw for interfacing with C routines.

8.38.3.7 RStrIter RC::RStr::end ( ) [inline]

Returns a bounds-safe iterator which points past the last element.

Accessing this iterator before decrementing it will trigger an ErrorMsgBounds. Accessing a decremented iterator after this RStr is destructed will throw ErrorMsgNull.

8.38.3.8 void RC::RStr::Get (f32 & *x*) const [inline]

Overloaded function to extract type f32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.9 void RC::RStr::Get (f64 & *x*) const [inline]

Overloaded function to extract type f64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.10 void RC::RStr::Get (f80 & *x*) const [inline]

Overloaded function to extract type f80 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.11 void RC::RStr::Get (u32 & *x*) const [inline]

Overloaded function to extract type u32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.12 void RC::RStr::Get (u64 & *x*) const [inline]

Overloaded function to extract type u64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.13 void RC::RStr::Get (i32 & *x*) const [inline]

Overloaded function to extract type i32 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.14 void RC::RStr::Get (i64 & *x*) const [inline]

Overloaded function to extract type i64 from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.15 void RC::RStr::Get (bool & *x*) const [inline]

Overloaded function to extract type bool from this class.

Parameters

x	The reference to which the value will be assigned.

8.38.3.16 f32 RC::RStr::Get_f32 ( ) const [inline]

Convert the beggining characters of this string to a float.

Note, these detect hexadecimal float numbers formatted like like 0xF.8 = 15.5, but they do not process octal.

8.38.3.17 u32 RC::RStr::Get_u32 (int *base* = 0) const [inline]

Convert the beggining characters of this string to this integer type.

Note, the default parameter for base autodetects according to decimal, unless 0-leading octal as 013 = 11, or 0x-leading hexadecimal as 0xF = 15.

8.38.3.18 bool RC::RStr::GetLine (std::istream & *in* = std::cin, char *delim* = '\n') [inline]

Set the string equal to the next line from istream in, up to character delim.

Usage example: RStr line; while(line.GetLine()) { cout << line << endl; } Note: Due to buffering, it can be more efficient to use: FileRead fr(stdin); while(fr.Get(line)) { ... } But GetLine is the better choice to avoid reading ahead, or for interactive input.

Returns

True if input was received.

8.38.3.19 RStr& RC::RStr::insert (size_t *pos_this*, const RStr & *s*, size_t *pos_s*, size_t *n*) [inline]

Inserts at index pos_this, n characters from s starting at pos_s.

Throws ErrorMsgBounds if pos_this or pos_s are out of their respective bounds. Inserts fewer characters if n is greater than the remaining characters.

8.38.3.20 RStr& RC::RStr::insert (size_t *pos_this*, const char * *s*, size_t *n*) [inline]

Inserts at index pos_this, n characters from s or until NULL-termination.

If s is NULL, nothing is inserted.

8.38.3.21 bool RC::RStr::Is_f32 (bool *strict* = false) const [inline]

True if the start of the string can be converted to an f32.

Parameters

strict	Return true only if the full string converts.

8.38.3.22 bool RC::RStr::Is_f64 (bool *strict* = `false`) const [inline]

True if the start of the string can be converted to an f64.

Parameters

strict	Return true only if the full string converts.

8.38.3.23 bool RC::RStr::Is_f80 (bool *strict* = `false`) const [inline]

True if the start of the string can be converted to an f80.

Parameters

strict	Return true only if the full string converts.

8.38.3.24 bool RC::RStr::Is_hex32 (bool *strict* = `false`) const [inline]

True if the start of the string can be converted as hexadecimal to a u32.

Parameters

strict	Return true only if the full string converts.

8.38.3.25 bool RC::RStr::Is_hex64 (bool *strict* = `false`) const [inline]

True if the start of the string can be converted as hexadecimal to a u64.

Parameters

strict	Return true only if the full string converts.

8.38.3.26 bool RC::RStr::Is_i32 (int *base* = `0`, bool *strict* = `false`) const [inline]

True if the start of the string can be converted to an i32.

Parameters

base	The conversion base to test (0 is automatic).
strict	Return true only if the full string converts.

8.38.3.27 bool RC::RStr::Is_i64 (int *base* = 0, bool *strict* = false) const [inline]

True if the start of the string can be converted to an i64.

Parameters

base	The conversion base to test (0 is automatic).
strict	Return true only if the full string converts.

8.38.3.28 bool RC::RStr::Is_u32 (int *base* = 0, bool *strict* = false) const [inline]

True if the start of the string can be converted to an u32.

Parameters

base	The conversion base to test (0 is automatic).
strict	Return true only if the full string converts.

8.38.3.29 bool RC::RStr::Is_u64 (int *base* = 0, bool *strict* = false) const [inline]

True if the start of the can be converted to an u64.

Parameters

base	The conversion base to test (0 is automatic).
strict	Return true only if the full string converts.

8.38.3.30 RStr RC::RStr::ISOtoUTF8 ( ) const [inline]

Treat this string as ISO-8859-1 and return a UTF8 string.

This supports the windows-1252 extension as required by HTML5.

8.38.3.31 template<class T > static RStr RC::RStr::Join (const Data1D< T > & *str_arr*, const RStr & *spacer* = " ") [inline], [static]

Takes a Data1D<T> array and joins them as one string with spacer between each element.

Note: Each element is converted to a string with its RStr(T) constructor, and thus only works on types for which a constructor exists.

8.38.3.32 size_t RC::RStr::Length8 ( ) const [inline]

Determine the length of the contents treated as a UTF-8 string.

Throws ErrorMsg "Invalid UTF-8" if the string cannot be parsed as a UTF-8 string.

8.38.3.33 template<class T > static RStr RC::RStr::MakeCSV (const Data1D< T > & *arr*) [inline], [static]

Makes a comma-separated values string out of a Data1D<T> arr.

The type must have an RStr(T) constructor. The elements are separated by ", ".

8.38.3.34 template<class T > static RStr RC::RStr::MakeCSV (const Data2D< T > & *arr*) [inline], [static]

Makes a comma-separated values string out of a Data2D<T> arr.

The elements are separated by ", " in the first dimension, and "\n" in the second dimension.

See also

MakeCSV

8.38.3.35 template<class T > static RStr RC::RStr::MakeCSV (const Data3D< T > & *arr*) [inline], [static]

Makes a comma-separated values string out of a Data3D<T> arr.

The elements are separated by ", " in the first dimension, "\n" in the second dimension, and "\n\n" in the third dimension.

See also

MakeCSV

8.38.3.36 Data1D<RStr> RC::RStr::SplitAny (const RStr & *dividers*) const [inline]

Return an array of strings split by any characters in dividers.

Always returns at least one element.

8.38.3.37 const char* RC::RStr::ToLPCTSTR ( ) [inline]

For Win32, provides an LPCTSTR to the string.

See ToLPCWSTR() for the return type if UNICODE is defined.

8.38.3.38 HoldRelated<std::wstring, const wchar_t*> RC::RStr::ToLPCWSTR ( ) [inline]

For Win32, provides an LPCWSTR to the string.

Return type implicitly casts to const whcar_t*, or with Get().

8.38.3.39 Data1D<u32> RC::RStr::UTF8toUTF32 ( ) const [inline]

Treat the contents as a UTF-8 string, and return corresponding UTF-32 data.

Throws ErrorMsg "Invalid UTF-8" if the string cannot be parsed as a UTF-8 string.

The documentation for this class was generated from the following file:

- RStr.h

8.39 RC::Segfault Class Reference

A static class for catching and throwing segfaults.

`#include <Errors.h>`

Static Public Member Functions

- static void Handler (int)

  The default handler for a segfault. It throws ErrorMsgFatal.
- static void SetHandler ()

  Call this to set the segfault handler to Segfault::Handler.

8.39.1 Detailed Description

A static class for catching and throwing segfaults.

Call Segfault::SetHandler() at the beginning of the program, and segfaults will result in an ErrorMsgFatal with a descriptive stack trace. In gcc -fnon-call-exceptions is required for the exception to propagate out of this handler. Note: This is automatically called with the RC_MAIN { } macro.

The documentation for this class was generated from the following file:

- Errors.h

8.40 RC::Sock Class Reference

A portable socket interface for reading and writing to an open socket.

`#include <Net.h>`

Public Member Functions

- Sock (SOCKET new_sock=INVALID_SOCKET)

  Encapsulate a new socket, receiving the base socket type of the system (int or SOCKET).
- void Close ()

  Manually close the socket.
- void SetRemote (RStr remote_addr, RStr remote_port)

  For record-keeping, set the remote address and port.
- RStr GetRemoteAddr () const

  Get the remote address this socket is connected to.
- RStr GetRemotePort () const

  Get the remote port this socket is connected to.
- bool DataReady (bool block_until_ready=false) const

  Returns true if data is ready for reading.
- bool CanSend (bool block_until_ready=false) const

  Returns true if the socket is ready to send data.
- FileRW ToFileRW ()

  On unix-based systems, return a FileRW coresponding to this socket.
- SOCKET Raw () const

  Return the raw encapsulated socket.
- template<class T >
  size_t Recv (Data1D< T > buf, bool do_block=true)

  Receive data from the socket into buf, up to the size of buf.
- size_t Recv (RStr &str, bool crop_newline=true, bool do_block=true)

  Receive characters into str until a newline or the end of data.
- template<class T >
  size_t Send (Data1D< T > buf, bool do_block=true)

  Sends the contents of buf through the socket.
- size_t Send (RStr str, bool do_block=true)

  Sends the contents of str throught he socket.

8.40.1 Detailed Description

A portable socket interface for reading and writing to an open socket.

This class shares its data in a reference-counted manner upon assignment or copy construction, and automatically closes the socket as the last instance goes out of scope.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 RC::Sock::Sock (SOCKET *new_sock* = INVALID_SOCKET) [inline]

Encapsulate a new socket, receiving the base socket type of the system (int or SOCKET).

Use Connect or Listener::Accept to initialize this.

8.40.3 Member Function Documentation

8.40.3.1 bool RC::Sock::CanSend (bool *block_until_ready* = `false`) const [inline]

Returns true if the socket is ready to send data.

If block_until_ready is true this will wait until data can be sent or an error occurs, such as the socket closing or a signal being received.

8.40.3.2 void RC::Sock::Close ( ) [inline]

Manually close the socket.

Note this is handled automatically when all copies of the Sock go out of scope.)

8.40.3.3 bool RC::Sock::DataReady (bool *block_until_ready* = `false`) const [inline]

Returns true if data is ready for reading.

If block_until_ready is true this will wait until data is available or an error occurs, such as the socket closing or a signal being received. (Note that rare kernel level events can cause data to no longer be available after it was reported as available.)

8.40.3.4 template<class T > size_t RC::Sock::Recv (Data1D< T > *buf,* bool *do_block* = `true`) [inline]

Receive data from the socket into buf, up to the size of buf.

Parameters

buf	The buffer into which data is received.
do_block	If true, wait until some data is available.

Returns

The number of elements of buf received.

8.40.3.5 size_t RC::Sock::Recv (RStr & *str,* bool *crop_newline* = `true`, bool *do_block* = `true`) [inline]

Receive characters into str until a newline or the end of data.

Parameters

str	The string into which a line is received.
crop_newline	If true, removes "\r\n" and "\n" from the end.
do_block	If true, keeps reading until newline or the socket closes. If false, reads until newline or no data is available.

Returns

The number of bytes received, including null characters and newlines not added to str.

8.40.3.6 template<class T > size_t RC::Sock::Send (Data1D< T > *buf,* bool *do_block* = `true`) [inline]

Sends the contents of buf through the socket.

Parameters

buf	The buffer containing the data to send.
do_block	If true, sends all of buf. If false, sends as much of buf as can be sent.

Returns

The amount of data sent.

8.40.3.7 size_t RC::Sock::Send (RStr *str,* bool *do_block* = `true`) [inline]

Sends the contents of str throught he socket.

Parameters

str	The string containing the data to send.
do_block	If true, sends all of str. If false, sends as much of str as can be sent.

Returns

The amount of data sent.

8.40.3.8 FileRW RC::Sock::ToFileRW ( ) [inline]

On unix-based systems, return a FileRW coresponding to this socket.

Throws ErrorMsgNet if it cannot convert the socket.

The documentation for this class was generated from the following file:

- Net.h

8.41 RC::Time Class Reference

Accesses and formats the system date and time, and provides high precision timing.

`#include <RTime.h>`

Public Member Functions

- void Start ()

  Start the timer.
- Time ()

  Default constructor. Starts the timer.
- f64 SinceStart ()

  Returns seconds since start in real time.
- f64 ProcSinceStart ()

  Returns seconds since start in process time.

Static Public Member Functions

- static f64 Get ()

  Returns seconds since 1970-01-01 epoch to a precision between 100 nanoseconds and 1 microsecond.
- static f64 Get (time_t tt)

  Converts a time_t value into seconds from the 1970-01-01 epoch.
- static f64 Get (struct tm tmval)

  Returns the seconds from the 1970-01-01 epoch for tmval.
- static f64 Get_UTC (struct tm tmval)

  Given a UTC struct tm, returns the seconds since epoch portably.
- static f64 GetPrecision ()

  Empirically determines the precision of Get(), and stores this value for future calls to this.
- static i32 GetTimezone ()

  Returns the number of seconds offset from UTC.
- static Data1D< i32 > GetTimezoneData ()

  Returns { hours, min, sec } offset from UTC.
- static struct tm Local2UTC (struct tm tmval)

  Returns the local time given UTC time tmval.
- static struct tm UTC2Local (struct tm tmval)

  Returns the UTC time given local time tmval.
- static time_t Get_time_t ()

  Returns time_t as number of seconds since 1970-01-01 00:00:00 UTC.
- static time_t Get_time_t (f64 val)

  Portably converts val as seconds since 1970-01-01 epoch to time_t.
- static time_t Get_time_t (struct tm tmval)

  Converts tmval in local timezone to time_t.
- static time_t Get_time_t_UTC (struct tm tmval)

  Converts tmval in UTC to time_t.
- static void Normalize (struct tm &tmval)

  Normalizes the input tmval.
- static struct tm Get_tm ()

  Provides a struct tm in the local timezone for the current time.
- static struct tm Get_tm (time_t tt)

  Provides a struct tm in the current timezone for the given time_t.
- static struct tm Get_tm (f64 val)

  Provides a struct tm in the local timezone for val seconds from the 1970-01-01 epoch.
- static struct tm Get_tm_UTC ()

  Provides a struct tm in UTC for the current time.
- static struct tm Get_tm_UTC (time_t tt)

Provides a struct tm in UTC for the given time_t.

- static struct tm Get_tm_UTC (f64 val)

  Provides a struct tm in the local timezone for val seconds from the 1970-01-01 epoch.
- static struct tm Get_tm (struct tm tmval)

  Returns a normalized struct tm.
- static RStr GetStr ()

  Provides the current time in the format "Fri Aug 1 17:05:25 2014".
- static RStr GetStr (struct tm tmval)

  Provides the time in the format "Fri Aug 1 17:05:25 2014".
- static RStr GetStr_UTC ()

  Provides the UTC time in the format "Fri Aug 1 17:05:25 2014".
- static RStr GetStr (const RStr &format)

  Provides the local time as a string with the format processed by strftime.
- static RStr GetStr (const RStr &format, struct tm tmval)

  Provides the tmval as a string with the format processed by strftime.
- static RStr GetStr_UTC (const RStr &format)

  Provides the UTC time as a string with the format processed by strftime.
- static RStr GetDate ()

  Returns a local date string formatted like "2011-07-28".
- static RStr GetDate (struct tm tmval)

  Returns a date string for tmval formatted like "2011-07-28".
- static RStr GetDate_UTC ()

  Returns a UTC date string formatted like "2011-07-28".
- static RStr GetTime ()

  Returns a local time string formatted like "19:49:18".
- static RStr GetTime (struct tm tmval)

  Returns a time string for tmval formatted like "19:49:18".
- static RStr GetTime_UTC ()

  Returns a UTC time string formatted like "19:49:18".
- static RStr GetDateTime ()

  Returns a local date-time string formatted like "2011-07-28_19-49-18", which is suitable for filenames.
- static RStr GetDateTime (struct tm tmval)

  Returns a date-time string from tmval formatted like "2011-07-28_19-49-18", which is suitable for filenames.
- static RStr GetDateTime_UTC ()

  Returns a UTC date-time string formatted like "2011-07-28_19-49-18", which is suitable for filenames.
- static struct tm ParseDate (const RStr &date, struct tm tmval)

  Parses a date string like 2011-07-28, 2011-7-28, 07-28, 7-28, or "/" instead of "-", into base tmval.
- static struct tm ParseDate (const RStr &date)

  Parses a date string like 2011-07-28, 2011-7-28, 07-28, 7-28, or "/" instead of "-", into base local time.
- static struct tm ParseDate_UTC (const RStr &date)

  Parses a date string like 2011-07-28, 2011-7-28, 07-28, 7-28, or "/" instead of "-", into base UTC time.
- static struct tm ParseTime (const RStr &date, struct tm tmval)

  Parses a time string like 19:49:18, 19:49, or "-" instead of ":", into base tmval.
- static struct tm ParseTime (const RStr &date)

  Parses a time string like 19:49:18, 19:49, or "-" instead of ":", into base local time.
- static struct tm ParseTime_UTC (const RStr &date)

  Parses a time string like 19:49:18, 19:49, or "-" instead of ":", into base UTC time.
- static struct tm ParseDateTime (const RStr &date, struct tm tmval)

  Parses a string like ParseDate and ParseTime separated by "_" or " ", into base tmval.
- static struct tm ParseDateTime (const RStr &date)

  Parses a string like ParseDate and ParseTime separated by "_" or " ", into base local time.

- static struct tm ParseDateTime_UTC (const RStr &date)

  Parses a string like ParseDate and ParseTime separated by "_" or " ", into base UTC time.
- static void Sleep (f64 seconds)

  Sleeps for a floating point number of seconds, with sub-second precision to the limit of the system.

8.41.1 Detailed Description

Accesses and formats the system date and time, and provides high precision timing.

8.41.2 Member Function Documentation

8.41.2.1 static f64 RC::Time::GetPrecision ( ) [inline],[static]

Empirically determines the precision of Get(), and stores this value for future calls to this.

Returns

The smallest amount by which Get() can increment.

The documentation for this class was generated from the following file:

- RTime.h

8.42 RC::TimeOfDay Class Reference

A class which obeys periodic time-of-day boundaries, and can manage times being within a daily time frame.

`#include <RTime.h>`

Public Member Functions

- TimeOfDay ()

  Default constructor, initializes to local time.
- TimeOfDay (struct tm tmval)

  Initializes to tmval.
- TimeOfDay (u32 hr, u32 min=0, u32 sec=0)

  Initializes to the given hr, min, sec from midnight.
- TimeOfDay & operator= (struct tm tmval)

  Initializes to tmval.
- TimeOfDay & operator-= (const TimeOfDay &other)

  Subtracts another TimeOfDay amount, obeying periodic boundaries.
- TimeOfDay operator- (const TimeOfDay &other) const

  Subtracts two TimeOfDay amounts, obeying periodic boundaries.
- TimeOfDay & operator+= (const TimeOfDay &other)

  Adds another TimeOfDay amount, obeying periodic boundaries.
- TimeOfDay operator+ (const TimeOfDay &other) const

Adds two TimeOfDay amounts, obeying periodic boundaries.

- bool operator< (const TimeOfDay &other) const

  True if this TimeOfDay is < the other.
- bool operator<= (const TimeOfDay &other) const

  True if this TimeOfDay is <= the other.
- bool operator> (const TimeOfDay &other) const

  True if this TimeOfDay is > the other.
- bool operator>= (const TimeOfDay &other) const

  True if this TimeOfDay is >= the other.
- bool operator== (const TimeOfDay &other) const

  True if this TimeOfDay is == the other.
- bool Between (const TimeOfDay &left, const TimeOfDay &right) const

  True if this TimeOfDay is later than left and earlier than right, obeying periodic boundaries.
- u32 Hr () const

  Returns the 0-23 hour value after midnight.
- u32 Min () const

  Returns the 0-59 minute value after midnight.
- u32 Sec () const

  Returns the 0-59 second value after midnight.
- u32 AsSeconds () const

  Returns the total seconds after midnight.
- u32 AsMinutes () const

  Returns the total minutes after midnight.
- u32 AsHours () const

  Returns the total hours after midnight.
- void FromSeconds (u32 newsec)

  Initializes from a number of seconds after midnight.
- void FromMinutes (u32 newmin)

  Initializes from a number of minutes after midnight.

8.42.1 Detailed Description

A class which obeys periodic time-of-day boundaries, and can manage times being within a daily time frame.

The documentation for this class was generated from the following file:

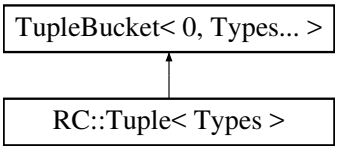
- RTime.h

8.43 RC::Tuple< Types > Class Template Reference

An efficient Tuple class with Set, Get, and an Apply function to pass the tuple contents on to any function.

`#include <Tuple.h>`

Inheritance diagram for RC::Tuple< Types >:



Public Member Functions

- Tuple ()
  Default constructor, elements use default constructors.
- template<int ResolveAmbiguity = 0>
  Tuple (const typename std::add_lvalue_reference< typename std::remove_reference< Types >::type >::type...types)
  Construct by const reference.
- template<bool ResolveAmbiguity = 0>
  Tuple (typename std::add_rvalue_reference< typename std::remove_reference< Types >::type >::type...types)
  Construct by moving elements in.
- std::tuple< Types... > GetStdTuple ()
  Returns a std::tuple of the same contents.
- template<size_t N>
  auto Get () -> decltype(this->GetHelper< N >(*this))
  Return element N in the Tuple.
- template<size_t N>
  auto Get () const -> decltype(this->GetHelper< N >(*this))
  Return element N in the Tuple.
- template<size_t N, class Val >
  void Set (Val val)
  Set element N in the Tuple.
- template<class Func >
  auto Apply (Func f) -> decltype(this->ApplyHelper(f,(typename SequenceGenerator< sizeof...(Types)>::Sequence())))
  Call function f with each element of the Tuple as an argument.
- template<class T >
  Data1D< T > AsData () const
  Return a Data1D array with each element constructed as type T.
- void Get (Types &...types)
  Get all elements at once, by reference.
- template<class... OtherTypes>
  Tuple< Types..., OtherTypes... > operator+ (Tuple< OtherTypes... > &other)
  Forms a new Tuple by concatenating two Tuples.

8.43.1 Detailed Description

template<class... Types>
class RC::Tuple< Types >

An efficient Tuple class with Set, Get, and an Apply function to pass the tuple contents on to any function.

8.43.2 Member Function Documentation

8.43.2.1 template<class... Types> template<class Func > auto RC::Tuple< Types >::Apply (Func *f*) -> decltype(this->ApplyHelper(f, (typename SequenceGenerator<sizeof...(Types)>::Sequence()))) [inline]

Call function f with each element of the Tuple as an argument.

Returns

The return value of function f for the given arguments.

The documentation for this class was generated from the following file:

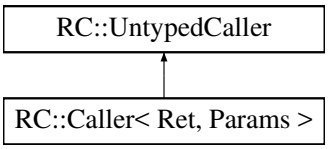
- Tuple.h

8.44 RC::UntypedCaller Class Reference

The base class of Caller without return type or parameters specified.

`#include <Caller.h>`

Inheritance diagram for RC::UntypedCaller:



8.44.1 Detailed Description

The base class of Caller without return type or parameters specified.

For managed casting to the correct Caller type use DynCaller

The documentation for this class was generated from the following file:

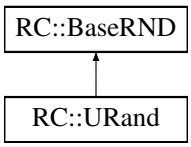
- Caller.h

8.45 RC::URand Class Reference

Cryptographically strong RNG, uses /dev/urandom.

`#include <RND.h>`

Inheritance diagram for RC::URand:



Public Member Functions

- URand (const size_t buf_size=128)
  Default constructor.
- virtual u32 Get_u32 ()
  Provides random u32 values.

Additional Inherited Members

8.45.1 Detailed Description

Cryptographically strong RNG, uses /dev/urandom.

The interface is identical to RND.h except there are no seed functions.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 RC::URand::URand (const size_t *buf_size* = 128) [inline]

Default constructor.

Parameters

buf_size	The number of u32 values to read at a time.

The documentation for this class was generated from the following file:

- RND.h

Chapter 9

File Documentation

9.1 APtr.h File Reference

A reference counting ptr that is auto-deleted as the last copy leaves scope.

```
#include "RCconfig.h"
#include "Types.h"
#include "Macros.h"
#include "Errors.h"
#include <iostream>
#include <atomic>
#include "PtrSharedCommon.h"
#include "PtrCommon.h"
```

Classes

- class RC::APtr< T >

  A reference counting ptr that auto-deletes what it points to when the last copy leaves scope or is otherwise destructed.

Namespaces

- RC

Functions

- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, APtr< T > obj)

  A convenience stream output for displaying the enclosed \ object.

9.1.1 Detailed Description

A reference counting ptr that is auto-deleted as the last copy leaves scope.

9.2 Bitfield.h File Reference

Provides a one-dimensional vector-like structure of packed bits.

```
#include "Data1D.h"
#include "Errors.h"
#include <iostream>
```

Classes

- class RC::Bitfield

  A bounds-safe one-dimensional vector-like structure of efficiently packed bits.
- class RC::Bitfield::BitfieldBool

  A temporary return-type that serves as an interface to specific bit values.
- class RC::Bitfield::BitfieldBoolConst

  A temporary return-type that serves as a const interface to specific bit values.

Namespaces

- RC

Functions

- std::ostream & RC::operator<< (std::ostream &out, const Bitfield &b)

  Outputs data to a stream as a character series of '1's and '0's.

9.2.1 Detailed Description

Provides a one-dimensional vector-like structure of packed bits.

9.3 Bitfield2D.h File Reference

Provides a two-dimensional structure of packed bits.

```
#include "Bitfield.h"
#include "Data1D.h"
#include "Errors.h"
#include <stdlib.h>
```

Classes

- class RC::Bitfield2D

  A bounds-safe two-dimensional structure of efficiently packed bits.

Namespaces

- RC

Functions

- std::ostream & RC::operator<< (std::ostream &out, const Bitfield2D &b)

  Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2.

9.3.1 Detailed Description

Provides a two-dimensional structure of packed bits.

9.4 Bitfield3D.h File Reference

Provides a three-dimensional structure of packed bits.

```
#include "Bitfield.h"
#include "Bitfield2D.h"
#include "Data1D.h"
#include "Errors.h"
#include <stdlib.h>
```

Classes

- class RC::Bitfield3D

  A bounds-safe three-dimensional structure of efficiently packed bits.

Namespaces

- RC

Functions

- std::ostream & RC::operator<< (std::ostream &out, const Bitfield3D &b)

  Outputs data to a stream as a character series of '1's and '0's, with newlines trailing each entry of dimension 2, and double-spaces after entries of dimension 3.

9.4.1 Detailed Description

Provides a three-dimensional structure of packed bits.

9.5 Caller.h File Reference

Provides a set of generalized functors for calling functions and methods.

```
#include "RCconfig.h"
#include "APtr.h"
#include "RevPtr.h"
#include <functional>
```

Classes

- class RC::UntypedCaller

  The base class of Caller without return type or parameters specified.
- class RC::Caller< Ret, Params >

  A general purpose function class which can refer to any static method, member method, functor, or lambda function.
- class RC::DynCaller

  A typeless container for a Caller, which has methods for dynamically casting it to the correct type.

Namespaces

- RC

Functions

- template<class C , class Ret , class... Params>
  Caller< Ret, Params... > RC::MakeCaller (C *obj, Ret(C::*func)(Params...))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class Ret , class... Params>
  Caller< Ret, Params... > RC::MakeCaller (C &obj, Ret(C::*func)(Params...))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class Ret , class... Params>
  Caller< Ret, C &, Params... > RC::MakeCaller (Ret(C::*func)(Params...))

  Generates a Caller for the member function with type inference, inserting a first parameter with a reference to the object.
- template<class Ret , class... Params>
  Caller< Ret, Params... > RC::MakeCaller (Ret(*func)(Params...))

  Generates a Caller for the specified static function, with type inference.
- template<class Ret , class... Params, class Functor >
  Caller< Ret, Params... > RC::MakeFunctor (Functor func)

  A special generator for functors, which requires specifying the return type and arguments.
- template<class C , class MemberFunc >
  auto RC::MakeCaller (Ptr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class MemberFunc >
  auto RC::MakeCaller (APtr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.
- template<class C , class MemberFunc >
  auto RC::MakeCaller (RevPtr< C > obj, MemberFunc func) -> decltype(MakeCaller(obj.Raw(), func))

  Generates a Caller for the member function of the given object, with type inference.

9.5.1 Detailed Description

Provides a set of generalized functors for calling functions and methods.

9.6 Data1D.h File Reference

Provides a one-dimensional vector-like structure.

```
#include "RCconfig.h"
#include "Errors.h"
#include "Iter.h"
#include "Macros.h"
#include "Types.h"
#include <algorithm>
#include <stdlib.h>
#include <iostream>
#include <type_traits>
```

Classes

- class RC::Data1D< T >

  A bounds-safe one-dimensional vector-like structure.

Namespaces

- RC

Functions

- std::ostream & RC::operator<< (std::ostream &out, const Data1D< char > &d)

  Outputs data to a stream as { a, b, ... }.
- std::ostream & RC::operator<< (std::ostream &out, const Data1D< u8 > &d)

  Outputs data to a stream as { 61, 62, ... }.
- std::ostream & RC::operator<< (std::ostream &out, const Data1D< i8 > &d)

  Outputs data to a stream as { 61, -42, ... }.
- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, const Data1D< T > &d)

  Outputs data to a stream as { elem0, elem1, ... }.
- template<class T >
  void RC::swap (RC::Data1D< T > &a, RC::Data1D< T > &b)

  Efficiently swap all the contents of a and b.
- template<class T , size_t N>
  auto RC::MakeData1D (T(&arr)[N]) -> RC::Data1D< T >

  Convenience generator for safely converting C-style arrays.

9.6.1 Detailed Description

Provides a one-dimensional vector-like structure.

9.7 Data2D.h File Reference

Provides a bounds-safe two-dimensional resizeable structure.

```
#include "Data1D.h"
#include "Errors.h"
#include <stdlib.h>
```

Classes

- class RC::Data2D< T >
  A bounds-safe two-dimensional resizeable structure.
- class RC::Data2D< T >
  A bounds-safe two-dimensional resizeable structure.

Namespaces

- RC

Functions

- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, const Data2D< T > &d)
  Outputs data to a stream as { { x_0_0, x_1_0, ...}, { x_0_1, x_1_1, ...}, ... }.
- template<class T2 >
  void RC::swap (Data2D< T2 > &a, Data2D< T2 > &b)
  Efficiently swap all the contents of a and b.

9.7.1 Detailed Description

Provides a bounds-safe two-dimensional resizeable structure.

9.8 Data3D.h File Reference

Provides a bounds-safe three-dimensional resizeable structure.

```
#include "Data1D.h"
#include "Data2D.h"
#include "Errors.h"
#include <stdlib.h>
```

Classes

- class RC::Data3D< T >

  A bounds-safe three-dimensional resizeable structure.
- class RC::Data3D< T >

  A bounds-safe three-dimensional resizeable structure.

Namespaces

- RC

Functions

- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, const Data3D< T > &d)

  Outputs data to a stream.
- template<class T2 >
  void RC::swap (Data3D< T2 > &a, Data3D< T2 > &b)

  Efficiently swap all the contents of a and b.

9.8.1 Detailed Description

Provides a bounds-safe three-dimensional resizeable structure.

9.9 Errors.h File Reference

Provides informative exception handling.

```
#include "RCconfig.h"
#include "Types.h"
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <typeinfo>
```

Classes

- class RC::ErrorMsg

  An exception class that records where the exception was thrown and provides a stack trace.
- class RC::ErrorMsgFatal

  A subtype of RC::ErrorMsg for Fatal errors.
- class RC::ErrorMsgNull

  A subtype of RC::ErrorMsg for Null errors.
- class RC::ErrorMsgBounds

  A subtype of RC::ErrorMsg for Bounds errors.
- class RC::ErrorMsgMemory

  A subtype of RC::ErrorMsg for Memory errors.
- class RC::ErrorMsgCast

  A subtype of RC::ErrorMsg for Bad Cast errors.
- class RC::ErrorMsgFile

  A subtype of RC::ErrorMsg for File related errors.
- class RC::ErrorMsgNet

  A subtype of RC::ErrorMsg for Networking related errors.
- class RC::Segfault

  A static class for catching and throwing segfaults.

Namespaces

- RC

Macros

- #define RC_MAKE_ERROR_TYPE(Type)

  Creates new RC::ErrorMsg subtypes.
- #define Throw_RC_Error(err) throw RC::ErrorMsg(err, __FILE__, __LINE__)

  Use this to throw an RC:ErrorMsg exception.
- #define Throw_RC_Type(Type, err) throw RC::ErrorMsg##Type(err, __FILE__, __LINE__)

  Use this to throw an RC:ErrorMsg subtype exception.
- #define Catch_RC_Error() catch (RC::ErrorMsgFatal& err) { fprintf(stderr, "Fatal Error: %s\n", err.what()); exit(-1); } catch (RC::ErrorMsg& err) { fprintf(stderr, "Error: %s\n", err.what()); }

  Place after a try block to catch RC errors and print the error text.
- #define Catch_RC_Error_Exit() catch (RC::ErrorMsg& err) { fprintf(stderr, "Error: %s\n", err.what()); exit(-1); }

  Place after a try block to catch RC errors, print the error text, and exit.

Variables

- const size_t RC::ErrorMsg_text_bufsize = 4096

  The maximum size of the char array returned by ErrorMsg::what() and ErrorMsg::GetError(), including the null.
- const size_t RC::ErrorMsg_type_bufsize = 256

  The maximum size of the char array returned by ErrorMsg::GetType(), including the null.

9.9.1 Detailed Description

Provides informative exception handling.

9.9.2 Macro Definition Documentation

9.9.2.1 #define Catch_RC_Error( ) catch (RC::ErrorMsgFatal& err) { fprintf(stderr, "Fatal Error: %s\n", err.what()); exit(-1); } catch (RC::ErrorMsg& err) { fprintf(stderr, "Error: %s\n", err.what()); }

Place after a try block to catch RC errors and print the error text.

Note: Exits on ErrorMsgFatal with return value -1.

9.9.2.2 #define RC_MAKE_ERROR_TYPE(*Type*)

Value:

```
class ErrorMsg##Type : virtual public RC::ErrorMsg {\
    public:\
    /** \brief The default constructor.  */ \
    /** Use the convenience macro Throw_RC_Type(Type, "Reason"); */ \
    ErrorMsg##Type(const char* new_err_msg, const char* filename = "",\
                   int line_number = 0)\
      : RC::ErrorMsg(new_err_msg, filename, line_number) {\
      snprintf(type_msg, ErrorMsg_type_bufsize, #Type);\
    }\
    /** \brief The destructor. */ \
    virtual ~ErrorMsg##Type() RC_NOEXCEPT {} \
    RC_DefaultCopyMove(ErrorMsg##Type) \
  };
```

Creates new RC::ErrorMsg subtypes.

Use as: "namespace RC { RC_MAKE_ERROR_TYPE(NewType) }" to make RC::ErrorMsgNewType

9.9.2.3 #define Throw_RC_Error(*err*) throw RC::ErrorMsg(err, __FILE__, __LINE__)

Use this to throw an RC:ErrorMsg exception.

It automatically adds the source file name and line number.

Parameters

err	(const char*) The reason for the exception.

9.9.2.4 #define Throw_RC_Type(*Type, err*) throw RC::ErrorMsg##Type(err, __FILE__, __LINE__)

Use this to throw an RC:ErrorMsg subtype exception.

It automatically adds the source file name and line number.

Parameters

Type	The subtype. e.g. Throw_RC_Type(Null, "Reason") for ErrorMsgNull.
err	(const char*) The reason for the exception.

9.10 File.h File Reference

Provides file input and output, and file / directory tools.

```
#include "RCconfig.h"
#include "Types.h"
#include "Errors.h"
#include "Ptr.h"
#include "Data1D.h"
#include "RStr.h"
#include <winsock2.h>
#include <direct.h>
#include <windows.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <atomic>
#include <type_traits>
```

Classes

- class RC::FileBase

  Provides the common methods for the FileRead/FileWrite/FileRW classes.
- class RC::FileRead

  A file reading class that provides buffered and unbuffered access to files with support for non-POD classes.
- class RC::FileWrite

  A file writing class that provides buffered and unbuffered output to files with support for non-POD classes.
- class RC::FileRW

  A file class for both reading and writing that provides buffered and unbuffered output to files.
- class RC::File

  A class with static methods for file and directory info and manipulation.

Namespaces

- RC

Enumerations

- enum RC::WriteMode { **TRUNCATE** =0, **KEEP**, **APPEND**, **NEWONLY** }

  Valid write modes for a FileWrite or FileRW.

Functions

- template<class T >
  bool RC::FileGetWrapper (FileRead &fr, T &data)

  Overload this with a specialization to support Get on a custom type.
- template<class T >
  void RC::FilePutWrapper (FileWrite &fw, const T &data)

  Overload this with a specialization to support Put on a custom type.
- template<>
  bool RC::FileGetWrapper< std::string > (FileRead &fr, std::string &data)

  A specialization for handling std::string objects.
- template<>
  void RC::FilePutWrapper< std::string > (FileWrite &fw, const std::string &data)

  A specialization for handling std::string objects.

9.10.1 Detailed Description

Provides file input and output, and file / directory tools.

9.11 Iter.h File Reference

Provides a bounds-checked iterator that knows when its target was deleted.

```
#include "Types.h"
#include "Macros.h"
#include "RevPtr.h"
#include <iterator>
```

Classes

- class RC::RAIter< Cont, T >

  A bounds-checked iterator for random-access containers that knows when its target was deleted.

Namespaces

- RC

Functions

- template<class Cont , class T >
  RAIter< Cont, T > RC::operator+ (size_t x, const RAIter< Cont, T > &iter)

  Returns a new iterator with an offset incremented by x.

9.11.1 Detailed Description

Provides a bounds-checked iterator that knows when its target was deleted.

9.12 Macros.h File Reference

Provides a set of convenience macros for metaprogramming and debugging.

```
#include "RCconfig.h"
```

Namespaces

- RC

Macros

- #define RC_NOP(...) __VA_ARGS__

  Evaluates to whatever is passed in. Does no operation.
- #define RC_CAT(a, b) RC_CAT_HELP(a,b)

  Concatenates two tokens.
- #define RC_EMPTY(...) RC_EMPTY_HELP(RC_EMPTY_HELP2(__VA_ARGS__))

  Returns 1 if no arguments passed, 0 if arguments are passed.
- #define RC_ARGS_NUM(...) RC_ARGNUMCAT(RC_ARGS_EVAL,RC_EMPTY(__VA_ARGS__))(__VA_↩ ARGS__)

  Returns the number of arguments passed to the macro, from 0 to 63.
- #define RC_ARGS_EACH(Macro, ...) RC_EV(RC_ARGS_EACH_HELP(RC_EMPTY(__VA_ARGS__↩))(Macro,__VA_ARGS__))

  Does RC_ARGS_EACH(Macro,a,b,c) –> Macro(a)Macro(b)Macro(c), for any number of parameters up to 342 or compiler limits.
- #define RC_ARGS_BET(func, bet, ...) RC_ARGS_BET_HELP(func,bet,__VA_ARGS__)

  Does RC_ARGS_BET(Func,Between,a,b,c) –> Func(a)Between Func(b)Between Func(c)
- #define RC_ARGS_LIST(func, ...) RC_EV(RC_ARGS_LIST_FIRST(RC_EMPTY(__VA_ARGS__))(func,_↩ _VA_ARGS__))

  Does RC_ARGS_LIST(Func,a,b,c) –> Func(a),Func(b),Func(c)
- #define RC_ARGS_PAIR(odd, even, ...) RC_EV(RC_ARGS_PAIR_FIRST(RC_EMPTY(__VA_ARGS__↩))(odd,even,__VA_ARGS__))

  Does RC_ARGS_PAIR(Odd,Even,a,b,c,d) –> Odd(a)Even(b),Odd(c)Even(d)
- #define RC_ARGS_DECLIST(...) RC_ARGS_PAIR(RC_NOP,RC_NOP,__VA_ARGS__)

  Does "RC_ARGS_DECLIST(int,a,int,b)" –> "int a,int b".
- #define RC_ARGS_PARAMLIST(...) RC_ARGS_PAIR(RC_ARGS_EAT,RC_NOP,__VA_ARGS__)

  Does "RC_ARGS_PARAMLIST(int,a,int,b)" –> "a,b".
- #define RC_DEBOUT(...) RC_DEBOUT_STREAM << __FILE__ << ":" << __LINE__ RC_ARGS_EA↩ CH(RC_DEBOUT_HELP,__VA_ARGS__) << std::endl;

  Use RC_DEBOUT(var1, ..., varN) for a flushed stderr debug printout of variables.
- #define RC_UNUSED_PARAM(v) v __attribute__((unused))

  Use in function headers as void func(int RC_UNUSED_PARAM(x)) to suppress warnings about non-use.
- #define RC_DYNAMIC_LOAD_FUNC_RAW(FuncName, Library)

  Use as RC_DYNAMIC_LOAD_FUNC_RAW(FuncName,Library) to load FuncName from the dynamic library file Library.
- #define RC_CONSTWRAP(Ret, Func, ...) RC_CONSTWRAP_HELP(Ret,Func,__LINE__,__VA_ARGS__)

  Use to metaprogrammatically generate paired const and non-const getters.
- #define RC_GetTc(T)

  Generates wrappers for const Get_T functions.

- #define RC_GetT(T)

  Generates wrappers for Get_T functions.
- #define RC_DEFAULT_COMPARISON()

  Given == and <, generates !=, >, <=, and >=.
- #define RC_STREAM_RAWWRAP(Type)

  Provides stream output for a class via Raw.
- #define RC_DefaultCopyMove(ClassName)

  Provides explicit default copy and move constructors and assignment.
- #define RC_ForIndex(i, cont) for (decltype(cont.size()) i = 0; i < cont.size(); ++i)

  Creates i of type equivalent to cont.size(), and loops over [0,cont.size). Use like: RC_ForIndex(i, cont) { cout << cont[i]; }.
- #define RC_ForEach(e, cont) for(decltype(cont.size()) RC_ForEach_i=0; RC_ForEach_i<cont.size(); ++RC_ForEach_i)if(bool RC_ForEach_b=false);else for(auto&& e=cont[RC_ForEach_i];RC_ForEach_b^=1;)

  Provides element auto&& e derived from the cont operator[], and loops over [0,cont.size). Like a range-based for loop, but using operator[]. Use like: RC_ForEach(e, cont) { cout << e; }.
- #define RC_ForRange(i, start, past_end) for (decltype(past_end) i = start; i < past_end; ++i)

  Creates a for loop over i with type deduction from the past_end.
- #define RC_Repeat(times) for (decltype(times) RC_Repeat_i = 0; RC_Repeat_i < times; ++RC_Repeat_i)

  Repeats the following block the specified times, with automatic type deduction.

Functions

- template<class T >
  void RC::UnusedVar (const T &)

  Mark an unused variable to suppress warnings.

9.12.1 Detailed Description

Provides a set of convenience macros for metaprogramming and debugging.

9.12.2 Macro Definition Documentation

9.12.2.1 #define RC_ARGS_BET(*func, bet, ...*) RC_ARGS_BET_HELP(func,bet,__VA_ARGS__)

Does RC_ARGS_BET(Func,Between,a,b,c) –> Func(a)Between Func(b)Between Func(c)

For example: cout RC_ARGS_BET(<< int, << ", ", 1.4, 2.5, 3.6); –> cout << int(1.4) << ", " << int(2.5) << ", " << int(3.5);

See also

RC_ARGS_EACH

9.12.2.2 #define RC_ARGS_LIST(*func, ...*) RC_EV(RC_ARGS_LIST_FIRST(RC_EMPTY(__VA_ARGS__))(func,__VA_ARGS__))

Does RC_ARGS_LIST(Func,a,b,c) –> Func(a),Func(b),Func(c)

See also

RC_ARGS_EACH

9.12.2.3 #define RC_ARGS_PAIR(*odd, even, ...*) RC_EV(RC_ARGS_PAIR_FIRST(RC_EMPTY(__VA_ARGS__↩))(odd,even,__VA_ARGS__))

Does RC_ARGS_PAIR(Odd,Even,a,b,c,d) –> Odd(a)Even(b),Odd(c)Even(d)

See also

RC_ARGS_LIST

9.12.2.4 #define RC_CONSTWRAP(*Ret, Func, ...*) RC_CONSTWRAP_HELP(Ret,Func,__LINE__,__VA_ARGS__)

Use to metaprogrammatically generate paired const and non-const getters.

The first parameter is the return type, and the second the function name. Subsequent parameters are pairs of type and variable name for function parameters. Access member variable x with "self.x". E.g.: RC_CONSTWRAP(int&, MyFunc, size_t, index) { return self.array[index]; }

9.12.2.5 #define RC_DEBOUT(*...*) RC_DEBOUT_STREAM << __FILE__ << ":" << __LINE__ RC_ARGS_EACH(RC_DEBOUT_HELP,__VA_ARGS__) << std::endl;

Use RC_DEBOUT(var1, ..., varN) for a flushed stderr debug printout of variables.

This also prints the file name and line number, and can be used to mark when execution reaches a point with no variables as RC_DEBOUT(). The default destination stream of std::cerr can be changed by defining RC_DEBOU↩ T_STREAM in RCconfig.h or earlier.

9.12.2.6 #define RC_DEFAULT_COMPARISON( )

Value:

```
/** \brief True if not equal. */ \
    template<class AnyValidType> \
    inline bool operator!= (const AnyValidType& other) const { \
      return !((*this) == other); \
    } \
    /** \brief True if other is less than this object. */ \
    template<class AnyValidType> \
    inline bool operator> (const AnyValidType& other) const { \
      return other < (*this); \
    } \
    /** \brief True if other is not less than this object. */ \
    template<class AnyValidType> \
    inline bool operator<= (const AnyValidType& other) const { \
      return !(other < (*this)); \
    } \
    /** \brief True if this object is not less than other. */ \
    template<class AnyValidType> \
    inline bool operator>= (const AnyValidType& other) const { \
      return !((*this) < other); \
    }
```

Given == and <, generates !=, >, <=, and >=.

9.12.2.7 #define RC_DefaultCopyMove(*ClassName*)

Value:

```
/** \brief Default copy constructor. */ \
  ClassName(const ClassName&) = default; \
  /** \brief Default move constructor. */ \
  ClassName(ClassName&&) = default; \
  /** \brief Default copy assignment operator. */ \
  ClassName& operator=(const ClassName&) & = default; \
  /** \brief Default move assignment operator. */ \
  ClassName& operator=(ClassName&&) & = default;
```

Provides explicit default copy and move constructors and assignment.

9.12.2.8 #define RC_DYNAMIC_LOAD_FUNC_RAW(*FuncName, Library*)

Value:

```
void* FuncName##_library = dlopen(Library, RTLD_LAZY | RTLD_GLOBAL); \
  static decltype(&FuncName) FuncName = (FuncName##_library) ? \
    (decltype(FuncName))dlsym(FuncName##_library,#FuncName): 0;
```

Use as RC_DYNAMIC_LOAD_FUNC_RAW(FuncName,Library) to load FuncName from the dynamic library file Library.

This creates a static function pointer named FuncName in-place to provide access to the loaded library. As it overrides FuncName, the previous declaration of FuncName MUST be in an outer scope (outside of the function where this is called), like from a header. After this macro, call FuncName with the same syntax as you would the previously declared FuncName. This macro is defined for both Linux and Windows. In the event of an error, FuncName is null. This must be checked in the current version of this macro.

9.12.2.9 #define RC_GetT(*T*)

Value:

```
/** \brief Overloaded function to extract type T from this class.
        @param x The reference to which the value will be assigned. */ \
    inline void Get(T &x) { x = Get_##T(); }
```

Generates wrappers for Get_T functions.

9.12.2.10 #define RC_GetTc(*T*)

Value:

```
/** \brief Overloaded function to extract type T from this class.
        @param x The reference to which the value will be assigned. */ \
    inline void Get(T &x) const { x = Get_##T(); }
```

Generates wrappers for const Get_T functions.

9.12.2.11 #define RC_STREAM_RAWWRAP(*Type*)

Value:

```
/** \brief A convenience stream output for displaying the enclosed \
      object. */ \
  template <class T> \
  inline std::ostream& operator<< (std::ostream &out, Type<T> obj) { \
    return (out << obj.Raw()); \
  }
```

Provides stream output for a class via Raw.

9.13 Net.h File Reference

Provides basic cross-platform socket networking, with support for both blocking and non-blocking sending and receiving.

```
#include "Types.h"
#include "Errors.h"
#include "File.h"
#include "RStr.h"
#include "Data1D.h"
#include "APtr.h"
#include <string.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <sys/types.h>
```

Classes

- class RC::Sock

  A portable socket interface for reading and writing to an open socket.
- class RC::Net

  Provides both client and server sides of blocking TCP connections.
- class RC::Net::Listener

  Listens to the specified port for incoming TCP connections.

Namespaces

- RC

Macros

- #define RC_SockType SOCKET

  The standard socket type used in socket calls for a given system.
- #define RC_InvSock INVALID_SOCKET

  The value returned for an invalid socket.
- #define RC_SockErr SOCKET_ERROR

  The value returned for a socket error.
- #define RC_CloseSock closesocket

  The function for manually closing sockets.
- #define RC_WouldBlock WSAEWOULDBLOCK

  The return value for a socket call which would block.
- #define RC_SockErrorAt WSAGetLastError()

  The variable or function call for obtaining the last error.
- #define RC_SockReturn int

  The return type for send and recv calls.

9.13.1 Detailed Description

Provides basic cross-platform socket networking, with support for both blocking and non-blocking sending and receiving.

9.14 Ptr.h File Reference

Provides a safe pointer class which throws exceptions.

```
#include "Errors.h"
#include "Macros.h"
#include <iostream>
#include "PtrCommon.h"
```

Classes

- class RC::Ptr< T >

  A safe pointer class that throws an RC::ErrorMsgNull if a null dereference is attempted.

Namespaces

- RC

Functions

- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, Ptr< T > obj)

  A convenience stream output for displaying the enclosed \ object.

9.14.1 Detailed Description

Provides a safe pointer class which throws exceptions.

9.15 PtrCommon.h File Reference

Common components for the *Ptr.h files. Do not include this directly.

Functions

- void Assert () const

  Throws RC::ErrorMsgNull if the pointer is null.
- bool IsSet () const

  True if the pointer is non-NULL.
- bool IsNull () const

  True if the pointer is NULL.
- T & operator* ()

  Dereferences the pointer, or throws an exception if null.
- T * operator-> ()

  Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.
- operator T * () const

  Implicitly casts to the enclosed pointer, without null checking.
- template<class Type >
  Ptr< Type > Cast ()

  Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.
- template<class Type >
  const Ptr< Type > Cast () const

  Const version of Cast()
- template<class Type >
  bool CanCast () const

  True if it can dynamically cast to this type.
- template<class Type >
  Type & As ()

  Dynamically casts and dereferences to the type presented as a template parameter, or throws ErrorMsgCast if it fails.
- template<class Type >
  const Type & As () const

  Const version of As()

9.15.1 Detailed Description

Common components for the *Ptr.h files. Do not include this directly.

9.15.2 Function Documentation

9.15.2.1 template<class Type > const T & As ( ) [inline]

Dynamically casts and dereferences to the type presented as a template parameter, or throws ErrorMsgCast if it fails.

Const version of operator*()

Dereference as the default type.

See also

operator()

9.15.2.2 template<class Type > Ptr<Type> Cast ( ) [inline]

Dynamically casts to an RC::Ptr of the type used as a template parameter on this function.

The Ptr is NULL if the cast fails.

9.15.2.3 bool IsNull ( ) const [inline]

True if the pointer is NULL.

Returns

True if the pointer is NULL.

9.15.2.4 bool IsSet ( ) const [inline]

True if the pointer is non-NULL.

Returns

True if the pointer is non-NULL.

9.15.2.5 const T & operator* ( ) [inline]

Dereferences the pointer, or throws an exception if null.

Const version of operator*()

The exception thrown is RC::ErrorMsgNull.

Returns

A reference to the dereferenced object.

9.15.2.6 const T * operator-> ( ) [inline]

Provides access to the enclosed pointer, or throws RC::ErrorMsgNull if null.

Const version of operator->()

Returns

The enclosed pointer.

9.16 PtrSharedCommon.h File Reference

Common components for the shared *Ptr.h files. Do not include directly.

Functions

- T * Raw () const

  Returns a direct reference to the enclosed pointer.

9.16.1 Detailed Description

Common components for the shared *Ptr.h files. Do not include directly.

9.16.2 Function Documentation

9.16.2.1 T* Raw ( ) const [inline]

Returns a direct reference to the enclosed pointer.

The reference is read/write, so it can be used as a function parameter which updates the value of this pointer.

Returns

A reference to the enclosed pointer.

9.17 RC.h File Reference

The main RC Library include file, which includes all other components.

```
#include "RCconfig.h"
#include <winsock2.h>
#include "Macros.h"
#include "Types.h"
#include "Errors.h"
#include "RCBits.h"
#include "Iter.h"
#include "Data1D.h"
#include "Data2D.h"
#include "Data3D.h"
#include "Bitfield.h"
#include "Bitfield2D.h"
#include "Bitfield3D.h"
#include "Ptr.h"
#include "APtr.h"
#include "RevPtr.h"
#include "Tuple.h"
#include "Caller.h"
#include "RStr.h"
#include "RTime.h"
#include "File.h"
#include "RND.h"
#include "Net.h"
```

Macros

- #define RC_USE

  Place this shorthand macro after including RC.h to use namespaces RC and std.
- #define RC_MAIN

  This macro wraps int main so that command line arguments are placed into RC::Data1D<RC::RStr> args.

9.17.1 Detailed Description

The main RC Library include file, which includes all other components.

9.17.2 Macro Definition Documentation

9.17.2.1 #define RC_MAIN

Value:

```
int RC_main(RC::Data1D<RC::RStr> &args); \
                int main (int argc, char *argv[]) { \
                  RC::Segfault::SetHandler();\
                  RC::Data1D<RC::RStr> args = RC::RStr::Args(argc, argv); \
                  return RC_main(args); \
                } \
                int RC_main(RC::Data1D<RC::RStr> &
      RC_UNUSED_PARAM(args))
```

This macro wraps int main so that command line arguments are placed into RC::Data1D<RC::RStr> args.

Usually args[0] is the executable name, and 1 through args.size()-1 will be the command line parameters. Usage example: RC_MAIN { if (args.size()<2) {Help(); return -1;} Code(args[1]); Here(); return 0; }

9.17.2.2 #define RC_USE

Value:

```
using namespace RC; \
               using namespace std; \
               namespace std {namespace placeholders{}} \
               using namespace std::placeholders;
```

Place this shorthand macro after including RC.h to use namespaces RC and std.

9.18 RCBits.h File Reference

Provides short convenience classes and functions.

```
#include "RCconfig.h"
#include "Tuple.h"
#include <iostream>
#include <sstream>
#include <string>
```

Classes

- class RC::LoopIndex

  A size_t like integer class which automatically stays within its range.
- class RC::HoldRelated< Hold, Provide >

  Stores the value of type Hold while providing access via type Provide.
- class RC::DebugTrack< ClassTracking, stack_trace >

  Inherit this class to add construction, destruction, and assignment output tracking.

Namespaces

- RC

Functions

- template<class T >
  BetwCompare< T > RC::Betw (const T &x)

  Returns a comparator that can be used for range comparisons.
- template<class... Args>
  OneOfCompare< Args... > RC::OneOf (Args...args)

  Returns a comparator that can be used to check if something is in a list.
- template<class T >
  RangeHelper< T > RC::Range (const T &start, const T &past_the_end)

  Provides an iterator which dereferences to the iterated values [start,past_the_end).
- template<class T2 >
  auto RC::IndexOf (const T2 &cont) -> RangeHelper< decltype(cont.size())>

  Provides an iterator which dereferences to the indices of cont from [0,cont.size()).

9.18.1 Detailed Description

Provides short convenience classes and functions.

9.19 RCconfig.h File Reference

The version information and configuration settings for RC Lib.

Macros

- #define RC_VERSION 201607061602ul

  The date-time stamped version number for this release.
- #define CPP11

  Defined if C++11 features are available.
- #define CPP14

  Defined if C++14 features are available.
- #define RC_HAVE_QT

  Define this if Qt is available in this compile. (Default off)
- #define RC_HAVE_BOOST

  Define this if Boost is available. (Default off)
- #define RC_NO_STACKTRACE

  Define this to disable stacktracing. (Default off) Note that for linux g++ stacktraces, one should use the compiler option -rdynamic.
- #define RC_HAVE_URAND

  If defined, a /dev/urandom file is available on the platform.
- #define RC_DEV_URANDOM "/dev/urandom"

  Set to the full pathname of the /dev/urandom file to use.
- #define RC_DEBOUT_STREAM std::cerr

  The output stream to which RC_DEBOUT debugging messages should be sent.

9.19.1 Detailed Description

The version information and configuration settings for RC Lib.

9.20 RevPtr.h File Reference

A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.

```
#include "Types.h"
#include "Errors.h"
#include "Macros.h"
#include <iostream>
#include <atomic>
#include "PtrSharedCommon.h"
#include "PtrCommon.h"
```

Classes

- class RC::RevPtr< T >

  A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.

Namespaces

- RC

Functions

- template<class T >
  std::ostream & RC::operator<< (std::ostream &out, RevPtr< T > obj)

  A convenience stream output for displaying the enclosed \ object.

9.20.1 Detailed Description

A reference counting pointer that revokes (NULLs) all copies when one set to AutoRevoke(true) leaves scope.

9.21 RND.h File Reference

Provides random number generator classes.

```
#include "Macros.h"
#include "Types.h"
#include "RTime.h"
#include "Data1D.h"
#include "RStr.h"
#include "File.h"
#include <stdio.h>
```

Classes

- class RC::BaseRND

  An abstract class which provides functions for obtaining randomness in convenient forms.
- class RC::RND

  A Mersenne Twister random number generator class with integer, array, or time-based seeding.
- class RC::EntropyRND

  Provides true random numbers sourced from environmental noise.
- class RC::URand

  Cryptographically strong RNG, uses /dev/urandom.

Namespaces

- RC

Macros

- #define RC_MAKE_GET_RANGE(classname)

  Generates a family of RND_Get_Range functions.
- #define RC_MAKE_RND_GEN(classname)

  Generates a family of random generator singletons.

Functions

- u64 RC::RND_Get_Range (u64 range)

  Uses RND as a RandomNumberGenerator, e.g. for random_shuffle.
- u64 RC::EntropyRND_Get_Range (u64 range)

  Uses EntropyRND as a RandomNumberGenerator, e.g. for random_shuffle.
- u64 RC::URand_Get_Range (u64 range)

  Uses URand as a RandomNumberGenerator, e.g. for random_shuffle.
- RND & RC::RND_Gen ()

  This returns a singleton of RND .
- EntropyRND & RC::EntropyRND_Gen ()

  This returns a singleton of EntropyRND .
- URand & RC::URand_Gen ()

  This returns a singleton of URand .

9.21.1 Detailed Description

Provides random number generator classes.

9.21.2 Macro Definition Documentation

9.21.2.1 #define RC_MAKE_GET_RANGE(*classname*)

Value:

```
/** \brief Uses classname as a RandomNumberGenerator, e.g. for random_shuffle
      @param range The range of values that can be returned, starting with 0.
      @return A random value from 0 up to and including range-1.
   */\
  inline u64 classname##_Get_Range(u64 range) {\
    static RC_MINGW_2014_BUGFIX classname rng;\
    return rng.GetRange(range);\
  }
```

Generates a family of RND_Get_Range functions.

9.21.2.2 #define RC_MAKE_RND_GEN(*classname*)

Value:

```
/** \brief This returns a singleton of classname. */\
  inline classname& classname##_Gen() {\
    static RC_MINGW_2014_BUGFIX classname rng;\
    return rng;\
  }
```

Generates a family of random generator singletons.

9.22 RStr.h File Reference

Provides a robust value-added wrapper for std::string.

```
#include "RCconfig.h"
#include "Macros.h"
#include "Types.h"
#include "Errors.h"
#include "RCBits.h"
#include "Data1D.h"
#include "Data2D.h"
#include "Data3D.h"
#include "Iter.h"
#include <tchar.h>
#include <algorithm>
#include <ctype.h>
#include <errno.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sstream>
#include <QString>
```

Classes

- class RC::RStr

  A bounds-safe string class which provides an identical interface to std::string plus many convenience functions for string manipulation.
- class RC::PluralStr

  Provides number-based singular/plural string management.

Namespaces

- RC

Macros

Convert to and from other supported types

- #define **RC_RSTR_Int_Input**(TYPE)
- #define **RC_RSTR_Float_Input**(TYPE)

Typedefs

- typedef RAIter< RStr, char > RC::RStrIter

  The random-access iterator for RStr begin() and end()

Enumerations

- enum RC::RStr_IntStyle {
  DEC =100000, **HEX**, **HEX0x**, **OCT**,
  OCT0, **BIN**, **CHAR** }
  The styles in which an integral number can be formatted.
- enum RC::RStr_FloatStyle { **AUTO** =100000, **FIXED**, **SCI** }
  The styles in which a floating point number can be formatted.

Functions

- RStr RC::operator+ (const RStr &lhs, const RStr &rhs)
  Concatenates two RStr'ings.
- std::istream & RC::operator>> (std::istream &in, RStr &rstr)
  Input text from a std::istream (e.g., std::cin) into an RStr.
- std::ostream & RC::operator<< (std::ostream &out, const RStr &rstr)
  Output text from an RStr to a std::ostream (e.g., std::cout).
- void RC::swap (RStr &lhs, RStr &rhs)
  Swaps two RStr'ings.
- std::istream & RC::getline (std::istream &is, RStr &str, char delim='\n')
  Sets str equal to one line from the input stream, up to end of file or the delim, which is discarded.
- bool RC::operator== (const RStr &lhs, const RStr &rhs)
  True if lhs equals rhs.
- bool RC::operator!= (const RStr &lhs, const RStr &rhs)
  True if lhs does not equal rhs.
- bool RC::operator< (const RStr &lhs, const RStr &rhs)
  True if lhs is less than rhs.
- bool RC::operator> (const RStr &lhs, const RStr &rhs)
  True if lhs is greater than rhs.
- bool RC::operator<= (const RStr &lhs, const RStr &rhs)
  True if lhs is less than or equal to rhs.
- bool RC::operator>= (const RStr &lhs, const RStr &rhs)
  True if lhs is greater than or equal to rhs.
- const RStr RC::REG_FLT ("[-+]?[0-9]*\\.?[0-9]+[eE][-+]?[0-9]+|[-+]?[0-9]*\\.?[0-9]+")
  A regular expression component to match a floating point number.
- const RStr RC::REG_FLTP ("([-+]?[0-9]*\\.?[0-9]+[eE][-+]?[0-9]+|[-+]?[0-9]*\\.?[0-9]+)")
  A regular expression component to match and return a floating point number.

9.22.1 Detailed Description

Provides a robust value-added wrapper for std::string.

9.22.2 Macro Definition Documentation

9.22.2.1 #define RC_RSTR_Float_Input(*TYPE*)

Value:

```
/** \brief Formats x as a string in the given style, and with it rounded
        to precision digits.
        \details For SCI precision is the significant digits to show, for
        AUTO it is the  most signicant digits to show before removing trailing
        zeroes, and for FIXED, precision is digits after the decimal.  AUTO
        has no exponent for numbers in the range (1e-5, 10^Precision). */ \
    inline RStr(TYPE x, RStr_FloatStyle style=AUTO, \
                u32 precision=std::numeric_limits<TYPE>::digits10) { \
      ParseFloat(x, style, precision); \
    }
```

9.22.2.2 #define RC_RSTR_Int_Input(*TYPE*)

Value:

```
/** \brief Formats x as a string in the given style, and with at least \
      precision 0-padded digits. */ \
    inline RStr(TYPE x, RStr_IntStyle style=DEC, i32 precision=-1) { \
      ParseInt(x, style, precision); \
    }
```

9.23 RTime.h File Reference

Provides classes for accessing and working with dates and times.

```
#include "Types.h"
#include "RStr.h"
#include "Data1D.h"
#include <ctime>
#include <Windows.h>
#include <Winbase.h>
```

Classes

- class RC::Time

  Accesses and formats the system date and time, and provides high precision timing.
- class RC::TimeOfDay

  A class which obeys periodic time-of-day boundaries, and can manage times being within a daily time frame.

Namespaces

- RC

Macros

- #define TIMEOFDAY_COMP(OP)

  Creates TimeOfDay comparitors.

9.23.1 Detailed Description

Provides classes for accessing and working with dates and times.

9.23.2 Macro Definition Documentation

9.23.2.1 #define TIMEOFDAY_COMP(*OP*)

Value:

```
/** \brief True if this TimeOfDay is OP the other. */ \
    inline bool operator OP \
    (const TimeOfDay& other) const {\
      return sec OP other.sec;\
    }
```

Creates TimeOfDay comparitors.

9.24 Tuple.h File Reference

Provides a Tuple class which can apply its contents as function parameters.

```
#include "RCconfig.h"
#include "Types.h"
#include "Data1D.h"
#include <tuple>
```

Classes

- class RC::Tuple< Types >

  An efficient Tuple class with Set, Get, and an Apply function to pass the tuple contents on to any function.

Namespaces

- RC

Functions

- template<class... Types>
  Tuple< Types... > RC::MakeTuple (Types...types)

  A convenience generator to make a Tuple from the given elements with type inference.

9.24.1 Detailed Description

Provides a Tuple class which can apply its contents as function parameters.

9.25 Types.h File Reference

Provides typedefs and routines for working with primitives.

```
#include "RCconfig.h"
#include "Macros.h"
#include <stdint.h>
#include <stddef.h>
#include <limits>
#include <cfloat>
#include <type_traits>
```

Classes

- class RC::Endian

  Auto-detects the endianness of the compilation target, and provides automatic endian conversion features.

Namespaces

- RC

Macros

- #define RC_HAVE_F80

  Defined if the 80-bit float was found available on this system.
- #define RC_THREAD_LOCAL thread_local

  Provides thread_local if available.
- #define RC_NOEXCEPT noexcept

  Provides noexcept if available.
- #define RC_TYPE_TRUE_MACRO(StructName, Type)

  For adding a new float or integral type, where StructName is FloatType or IntegerType.

Typedefs

- typedef int8_t i8

  8-bit signed integer.
- typedef uint8_t u8

  8-bit unsigned integer.
- typedef int16_t i16

  16-bit signed integer.
- typedef uint16_t u16

  16-bit unsigned integer.
- typedef int32_t i32

  32-bit signed integer.
- typedef uint32_t u32

  32-bit unsigned integer.
- typedef int64_t i64

  64-bit signed integer.

- typedef uint64_t u64

  64-bit unsigned integer.
- typedef float f32

  32-bit float.
- typedef double f64

  64-bit float.
- typedef long double f80

  80-bit float. (Note: Usually takes 16 bytes.)
- typedef f128 fBIGGEST

  The biggest float type available.
- template<class FT >
  using RC::FuncPtr = FT *

  Clean C-style function pointers: FuncPtr<void(int)> f = &func; f(4);.
- template<class C , class FT >
  using RC::MemFuncPtr = FT(C::*)

  Clean function pointer syntax for member function.

Functions

- template<class T >
  bool RC::IsSignedType (const T &x __attribute__((unused))=0)

  True if the type is signed.
- template<class T >
  bool RC::IsIntegerType (const T &x __attribute__((unused))=1)

  True if the type is an integer type.
- template<class T >
  void RC::AssertFloat ()

  For generating compilation errors if the type is not a float.
- template<class T >
  void RC::AssertInteger ()

  For generating compilation errors if the type is not an integer.
- template<class T >
  auto RC::ForceFloat (T val) -> typename ForceFloatHelper< T, FloatType< T >::value >::type

  If T is a float type, return the same type. Otherwise return it as an f64.
- template<class T >
  std::make_unsigned< T >::type RC::ForceUnsigned (T val)

  Returns the unsigned equivalent to the given type.
- template<class T >
  std::make_signed< T >::type RC::ForceSigned (T val)

  Returns the signed equivalent to the given type.
- template<class T >
  T RC::MIN_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::min())

  Provide the minimum value held by this type. For floats, the smallest positive value.
- template<class T >
  T RC::MAX_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::max())

  Provide the maximum value held by this type.
- template<class T >
  T RC::LOW_VAL (const T &x __attribute__((unused))=std::numeric_limits< T >::lowest())

  Provide the lowest value held by this type, for which no value is less.
- template<class T >
  T RC::MIN_POS (const T &x __attribute__((unused))=MIN_POS_Helper< T, FloatType< T >::value >::F())

  Provide the minimum positive value held by this type.

9.25.1 Detailed Description

Provides typedefs and routines for working with primitives.

9.25.2 Macro Definition Documentation

9.25.2.1 #define RC_TYPE_TRUE_MACRO(*StructName, Type*)

Value:

```
template<> struct StructName<Type> { static const bool value = true; };\
  template<> struct StructName<const Type> { static const bool value = true; };\
  template<> struct StructName<volatile Type> { static const bool value = true; };\
  template<> struct StructName<const volatile Type> { static const bool value = true; };
```

For adding a new float or integral type, where StructName is FloatType or IntegerType.

Index