CryptoGateway Documentation

Adrian Bedard Jonathan Bedard

May 19, 2016

Contents

Part I

CryptoGateway Library

Chapter 1

Introduction

The CryptoGateway library contains classes which handle cryptography. CryptoGateway is designed as an open source library, so much of the cryptography within the library is relatively simple. CryptoGateway is not meant to define cryptography to be used widely, rather, it is meant to provide a series of generalized hooks and interfaces which can be extended to various cryptographic algorithms.

1.1 Namespace

CryptoGateway uses the crypto namespace. The crypto namespace is designed for class, functions and constants related to cryptography. CrytpoGateway depends on many of the tools defined in the os namespace. Additionally, the crypto namespace contains a series of nested namespaces which help to disambiguate constants.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 binaryEncryption.cpp File Reference

Implementation of binary encryption files.

3.1.1 Detailed Description

Implementation of binary encryption files.

Author

Jonathan Bedard

Date

4/18/2016

Bug None

Implements the binary encryption files. Consult **binaryEncryption.h** (p. 7) for details on using these classes.

3.2 binaryEncryption.h File Reference

Definition of binary encryption files.

Classes

- class **crypto::binaryEncryptor**
  Encrypted binary file output.
- class **crypto::binaryDecryptor**
  Encrypted binary file output.

Namespaces

- **crypto**

3.2.1 Detailed Description

Definition of binary encryption files.

Author

Jonathan Bedard

Date

3/7/2016

Bug None

Provides an interface to dump and retrieve data from an encrypted binary file without concern as to the encryption algorithm used.

3.3 c_BaseTen.c File Reference

Implementation of base-10 algorithms.

3.3.1 Detailed Description

Implementation of base-10 algorithms.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements all of the basic functionality of a base-10 integer. All integer operations, both basic and otherwise, are implemented in this file.

3.4 c_BaseTen.h File Reference

Base-10 number functions.

Functions

- struct **numberType** ∗ **buildBaseTenType** ()

  Construct a base-10 number.
- int **base10Addition** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)

  Base-10 addition.
- int **base10Subtraction** (const uint32_t ∗src1, const uint32_t ∗src2, uint32_t ∗dest, uint16_t length)

Base-10 subtraction.

- int **base10Multiplication** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 multiplication.
- int **base10Division** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 division.
- int **base10Modulo** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 modulo.
- int **base10Exponentiation** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)

  Base-10 exponentiation.
- int **base10ModuloExponentiation** (const uint32_t *src1, const uint32_t *src2, const uint32_t *src3, uint32_t *dest, uint16_t length)
- int **base10GCD** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)
- int **base10ModInverse** (const uint32_t *src1, const uint32_t *src2, uint32_t *dest, uint16_t length)
- int **primeTest** (const uint32_t *src1, uint16_t test_iteration, uint16_t length)

3.4.1 Detailed Description

Base-10 number functions.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains functions which define a base-10 integer. There functions are bound to a number type.

3.4.2 Function Documentation

int base10Addition (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 addition.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Division (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 division.

This function takes in two arrays which represent base-10 numbers, preforms src1/src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Exponentiation (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 exponentiation.

This function takes in two arrays which represent base-10 numbers, preforms src1+src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10GCD (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

int base10ModInverse (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

int base10Modulo (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 modulo.

This function takes in two arrays which represent base-10 numbers, preforms src1src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10ModuloExponentiation (const uint32_t * src1, const uint32_t * src2, const uint32_t * src3, uint32_t * dest, uint16_t length)

int base10Multiplication (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 multiplication.

This function takes in two arrays which represent base-10 numbers, preforms src1*src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int base10Subtraction (const uint32_t * src1, const uint32_t * src2, uint32_t * dest, uint16_t length)

Base-10 subtraction.

This function takes in two arrays which represent base-10 numbers, preforms src1-src2 on the pair and then output the result to dest. Note that all three arrays must be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

struct **numberType*** buildBaseTenType ( )

Construct a base-10 number.

This function will return a **numberType** (p. 194) pointer defining the function pointers for a base-10 number. Note that the resulting pointer points to a structure which is static to the **c_BaseTen.c** (p. 8) file.

Returns

Pointer to **numberType** (p. 194) of type base-10

int primeTest (const uint32_t * src1, uint16_t test_iteration, uint16_t length)

3.5 c_cryptoTesting.cpp File Reference

Implementation for C file testing.

3.5.1 Detailed Description

Implementation for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file implements test suites which are testing raw C code. This file currently tests the Base-↩ Ten suite.

3.6 c_cryptoTesting.h File Reference

Header for C file testing.

3.6.1 Detailed Description

Header for C file testing.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This header is meant for the test suites which are testing raw C code. This header currently contains the Base-Ten suite.

3.7 c_numberDefinitions.c File Reference

Implementation of basic number.

3.7.1 Detailed Description

Implementation of basic number.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Most numerical operations must be defined by the specific number type, but a select few are generally applicable across all number types, these are implemented here.

3.8 c_numberDefinitions.h File Reference

Basic number declarations.

Classes

- struct **numberType**

  Number type function structure.

Typedefs

- typedef int(∗ **operatorFunction**) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

  Operator function typedef.
- typedef int(∗ **tripleCalculation**) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

  Triple operator function typedef.
- typedef int(∗ **shiftFunction**) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)

  Shift operator function typedef.
- typedef int(∗ **compareFunction**) (const uint32_t ∗, const uint32_t ∗, uint16_t)

  Comparison function typedef.

Functions

- struct **numberType** ∗ **buildNullNumberType** ()

  Construct a NULL number.
- int **standardCompare** (const uint32_t ∗src1, const uint32_t ∗src2, uint16_t length)

  Standard comparision.
- int **standardRightShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)

  Right shift.
- int **standardLeftShift** (const uint32_t ∗src1, uint16_t src2, uint32_t ∗dest, uint16_t length)

  Left shift.

3.8.1 Detailed Description

Basic number declarations.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Contains function typedefs used for various number operations and defines a few nearly universal numerical functions.

3.8.2 Typedef Documentation

typedef int(∗ compareFunction) (const uint32_t ∗, const uint32_t ∗, uint16_t)

Comparison function typedef.

This function typedef defines a function which takes in two arrays which represent numbers and then compares them.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint32_t*∗	Argument 2
in	*uint16_t*	size

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

typedef int(∗ operatorFunction) (const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Operator function typedef.

This function typedef defines a function which takes in two arrays which represent numbers, preform some operation on the pair and then output the result to a third array.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint32_t*∗	Argument 2
out	*uint32_t*∗	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

typedef int(∗ shiftFunction) (const uint32_t ∗, uint16_t, uint32_t ∗, uint16_t)

Shift operator function typedef.

This function typedef defines a function which takes in an array representing a number, shifts it the provided number of bits and outputs the result into the second array.

Parameters

in	*uint32_t*∗	Argument 1
in	*uint16_t*	Bits to shift
out	*uint32_t*∗	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

typedef int(∗ tripleCalculation) (const uint32_t ∗, const uint32_t ∗, const uint32_t ∗, uint32_t ∗, uint16_t)

Triple operator function typedef.

This function typedef defines a function which takes in three arrays which represent numbers, preform some operation on the triple and then output the result to a fourth array.

Parameters

in	*uint32_t**	Argument 1
in	*uint32_t**	Argument 2
in	*uint32_t**	Argument 3
out	*uint32_t**	Output
in	*uint16_t*	size

Returns

1 if success, 0 if failed

3.8.3 Function Documentation

struct **numberType*** buildNullNumberType ( )

Construct a NULL number.

This function will return a **numberType** (p. 194) pointer defining the function pointers for a NULL number. Note that the resulting pointer points to a structure which is static to the **c_number↩ Definitions.c** (p. 13) file.

Returns

Pointer to **numberType** (p. 194) of type NULL

int standardCompare (const uint32_t * src1, const uint32_t * src2, uint16_t length)

Standard comparision.

This function takes in two arrays which represent numbers and then compares them.

Parameters

in	*src1*	Argument 1
in	*src2*	Argument 2
in	*length*	Number of uint32_t in the arrays

Returns

-1 if 1<2, 0 if 1==2, 1 if 1>2

int standardLeftShift (const uint32_t * src1, uint16_t src2, uint32_t * dest, uint16_t length)

Left shift.

Shifts the bits in src1 in the left direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Bits to shift
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

int standardRightShift (const uint32_t * src1, uint16_t src2, uint32_t * dest, uint16_t length)

Right shift.

Shifts the bits in src1 in the right direction src2 number of bits. Output the result in dest. Note that dest and src1 should be the same size.

Parameters

in	*src1*	Argument 1
in	*src2*	Bits to shift
out	*dest*	Output
in	*length*	Number of uint32_t in the arrays

Returns

1 if success, 0 if failed

3.9 cryptoCConstants.h File Reference

Extern declarations of C constants.

Variables

- const int **crypto_numbertype_default**

  Default number ID.
- const int **crypto_numbertype_base10**

  Base-10 number ID.
- const char * **crypto_numbername_default**

  Default number marker.
- const char * **crypto_numbername_base10**

  Base-10 number marker.

3.9.1 Detailed Description

Extern declarations of C constants.
Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Declares a number of constants needed by both the C numerical algorithms and by C++ number classes.

3.9.2 Variable Documentation

const char* crypto_numbername_base10

Base-10 number marker.
This constant is "Base 10 Type". It represents a number of type base-10, or standard integer.

const char* crypto_numbername_default

Default number marker.
This constant is "NULL Type". It represents an untyped number.

const int crypto_numbertype_base10

Base-10 number ID.
This constant is 1. It represents a number of type base-10, or standard integer.

const int crypto_numbertype_default

Default number ID.
This constant is 0. It represents an untyped number.

3.10 cryptoCHeaders.h File Reference

Collected headers for C source code.

3.10.1 Detailed Description

Collected headers for C source code.
Author

Jonathan Bedard

Date

2/20/2016

Bug None

3.11 cryptoConstants.cpp File Reference

Implementation of CryptoGateway constants.

3.11.1 Detailed Description

Implementation of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Binds all of the scoped constants used by CryptoGateway. The nested namespaces ensure that there is no ambiguity as to the purpose and nature of the constants.

3.12 cryptoConstants.h File Reference

Extern definitions of CryptoGateway constants.

3.12.1 Detailed Description

Extern definitions of CryptoGateway constants.

Author

Jonathan Bedard

Date

3/19/2016

Bug None

Consult **cryptoConstants.cpp** (p. 19) for details. This file merely defines extern references to the global constants in **cryptoConstants.cpp** (p. 19).

3.13 cryptoCSource.cpp File Reference

Implementation of all C code.

3.13.1 Detailed Description

Implementation of all C code.

Author

Jonathan Bedard

Date

2/13/2016

Bug No known bugs.

This file includes all of the .c files needed for this library. It allows the CMake scripts for this project to be entirely C++ while still includeing raw C code.

3.14 cryptoError.cpp File Reference

Implementation of error sender and listener.

3.14.1 Detailed Description

Implementation of error sender and listener.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

Implements the error sender and listeners. These classes allow for managing the throwing of **crypto::errorPointer** (p. 52). Consult **cryptoError.h** (p. 20) for details.

3.15 cryptoError.h File Reference

Declaration of cryptographic errors.

Classes

- class **crypto::error**
  Sortable exception.
- class **crypto::passwordSmallError**
  Symmetric key too small.
- class **crypto::passwordLargeError**
  Symmetric key too big.

- class **crypto::bufferSmallError**

  Buffer too small.
- class **crypto::bufferLargeError**

  Buffer too large.
- class **crypto::insertionFailed**

  ADS Insertion Failed.
- class **crypto::customError**

  *Custom **crypto::error*** (p. 81)*.*
- class **crypto::fileOpenError**

  File open error.
- class **crypto::fileFormatError**

  File format error.
- class **crypto::illegalAlgorithmBind**

  Algorithm bound failure.
- class **crypto::hashCompareError**

  Hash mis-match.
- class **crypto::hashGenerationError**

  Hash generation error.
- class **crypto::actionOnFileError**

  File error.
- class **crypto::actionOnFileClosed**

  File closed error.
- class **crypto::publicKeySizeWrong**

  Public-key size error.
- class **crypto::keyMissing**

  Key missing error.
- class **crypto::NULLPublicKey**

  NULL public-key error.
- class **crypto::NULLDataError**

  NULL data error.
- class **crypto::NULLMaster**

  NULL master error.
- class **crypto::masterMismatch**

  Master mis-match.
- class **crypto::unknownErrorType**

  Unknown error.
- class **crypto::stringTooLarge**

  String size error.
- class **crypto::errorListener**

  crypto::error (p. 81) *listener*
- class **crypto::errorSender**

  *Sends **crypto::error*** (p. 81)*.*

Namespaces

- **crypto**

Typedefs

- typedef os::smart_ptr< error > **crypto::errorPointer**

  *Smart pointer to **crypto::error** (p. 81).*

3.15.1 Detailed Description

Declaration of cryptographic errors.

Author

Jonathan Bedard

Date

4/1/2016

Bug None

Declares a number of errors for the CryptoGateway package. Also declares two classes to manage the sending and listening for the throwing of **crypto::errorPointer** (p. 52).

3.16 cryptoFileTest.cpp File Reference

Implementation for cryptographic file testing.

3.16.1 Detailed Description

Implementation for cryptographic file testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file implements a series of tests designed to confirm the stability of cryptographic save file and load file functions.

3.17 cryptoFileTest.h File Reference

Header for cryptographic file testing.

3.17.1 Detailed Description

Header for cryptographic file testing.

Author

Jonathan Bedard

Date

3/5/2016

Bug No known bugs.

This contains a number of test suites and supporting classes which are designed to test the functionality of saving and loading cryptographic files, both binary and EXML.

3.18 CryptoGateway.h File Reference

Global include file.

Namespaces

- **crypto**

Variables

- bool **crypto::global_logging**
  Deprecated logging flag.

3.18.1 Detailed Description

Global include file.

Author

Jonathan Bedard

Date

4/16/2016

Bug None

This file contains all of the headers in the CryptoGateway library. Project which depend on the CryptoGateway library need only include this file.

3.19 cryptoHash.cpp File Reference

Implementation of crypto hashing.

3.19.1 Detailed Description

Implementation of crypto hashing.
Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements basic hashing frameworks and the XOR hash. Note that the XOR hash is not cryptographically secure. Consult **cryptoHash.h** (p. 24) for details.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Implements the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure. Consult the **RC4_Hash.h** (p. 37) for details.

3.20 cryptoHash.h File Reference

Declaration of crypto hashing.

Classes

- class **crypto::hash**
  Base hash class.
- class **crypto::xorHash**
  XOR hash class.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const hash &num)
  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, hash &num)
  Input stream operator.
- template<class hashClass >
  hashClass **crypto::hashData** (uint16_t hashType, const unsigned char *data, uint32_t length)
  Hashes data with the specified algorithm.

3.20.1 Detailed Description

Declaration of crypto hashing.
Implementation of RC4 hash.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares base cryptographic hashing class and functions. All hash algorithms should extend this hash class.

Author

Jonathan Bedard

Date

2/23/2016

Bug None

Declares the RC-4 hash algorithm. The RC-4 hashing algorithm is likely secure, but not proven secure.

3.21 cryptoLogging.cpp File Reference

Logging for crypto namespace, implementation.

3.21.1 Detailed Description

Logging for crypto namespace, implementation.
Jonathan Bedard

Date

2/23/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the crypto namespace.

3.22 cryptoLogging.h File Reference

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::cryptoout_func** ()
  Standard out object for crypto namespace.
- std::ostream & **crypto::cryptoerr_func** ()
  Standard error object for crypto namespace.

Variables

- os::smart_ptr< std::ostream > **crypto::cryptoout_ptr**
  Standard out pointer for crypto namespace.
- os::smart_ptr< std::ostream > **crypto::cryptoerr_ptr**
  Standard error pointer for crypto namespace.

3.23 cryptoNumber.cpp File Reference

Implements basic number types.

3.23.1 Detailed Description

Implements basic number types.

Author

Jonathan Bedard

Date

4/3/2016

Bug No known bugs.

Implements basic large numbers and the more specific large integer. Consult **cryptoNumber.h** (p. 26) for details.

3.24 cryptoNumber.h File Reference

Defines basic number types.

Classes

- class **crypto::number**

  Basic number definition.
- class **crypto::integer**

  Integer number definition.

Namespaces

- **crypto**

Functions

- std::ostream & **crypto::operator**<< (std::ostream &os, const number &num)

  Output stream operator.
- std::istream & **crypto::operator**>> (std::istream &is, number &num)

  Input stream operator.

3.24.1 Detailed Description

Defines basic number types.

Author

Jonathan Bedard

Date

3/2/2016

Bug No known bugs.

Contains declarations of large numbers for usage inside the CryptoGateway. The two numbers defined in this file are the general structure for large numbers and a basic integer.

3.25 cryptoNumberTest.cpp File Reference

Testing **crypto::number** (p. 177) and **crypto::integer** (p. 127).

3.25.1 Detailed Description

Testing **crypto::number** (p. 177) and **crypto::integer** (p. 127).

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file has a series of tests which confirm the functionality of **crypto::integer** (p. 127) and it's base class, **crypto::number** (p. 177).

3.26 cryptoPublicKey.cpp File Reference

Generalized and RSA public key implementation.

3.26.1 Detailed Description

Generalized and RSA public key implementation.

Author

Jonathan Bedard

Date

5/5/2016

Bug No known bugs.

Contains implementation of the generalized public key and the RSA public key. Consult **crypto↩ PublicKey.h** (p. 28) for details.

3.27 cryptoPublicKey.h File Reference

Generalized and RSA public keys.

Classes

- class **crypto::keyChangeReceiver**

  Interface for receiving key changes.
- class **crypto::keyChangeSender**

  *Interface inherited by **publicKey** (p. 199).*
- class **crypto::publicKey**

  Base public-key class.
- class **crypto::publicRSA**

  RSA public-key encryption.
- class **crypto::RSAKeyGenerator**

  Helper key generation class.

Namespaces

- **crypto**

3.27.1 Detailed Description

Generalized and RSA public keys.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Contains declarations of the generalized public key and the RSA public key. These classes can both encrypt and decrypt public keys.

3.28 cryptoTest.cpp File Reference

CryptoGateway library test constructor.

3.28.1 Detailed Description

CryptoGateway library test constructor.

Author

Jonathan Bedard

Date

4/7/2016

Bug No known bugs.

Binds all test suites for the test::CryptoGatewayLibraryTest. This library test is called "Crypto↩ Gateway."

3.29 cryptoTest.h File Reference

CryptoGateway library test header.

3.29.1 Detailed Description

CryptoGateway library test header.

Author

Jonathan Bedard

Date

4/2/2016

Bug No known bugs.

Contains declarations need to bind the CryptoGateway test library to the unit test driver.

3.30 gateway.cpp File Reference

Implements the gateway.

3.30.1 Detailed Description

Implements the gateway.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Implements the gateway defined in **gateway.h** (p. 30). Consult **gateway.h** (p. 30) for details.

3.31 gateway.h File Reference

Defines the gateway.

Classes

- class **crypto::gatewaySettings**
  Holds settings for gateway encryption.
- class **crypto::gateway**
  Security gateway.

Namespaces

- **crypto**

3.31.1 Detailed Description

Defines the gateway.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains the declaration for the gateway and the gateway settings. This header file is the culmination of the CryptoGateway library.

Note that due to development constraints, the gatewaySettings class is being pushed out in a frame-work form and is intended to contain a large set of algorithm definitions as well as an algorithm use agreement protocol.

3.32 gatewayTest.cpp File Reference

Implementation for end-to-end gateway testing.

3.32.1 Detailed Description

Implementation for end-to-end gateway testing.

Author

Jonathan Bedard

Date

4/26/2016

Bug No known bugs.

This file contains implementation of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

3.33 gatewayTest.h File Reference

Header for end-to-end gateway testing.

3.33.1 Detailed Description

Header for end-to-end gateway testing.

Author

Jonathan Bedard

Date

3/20/2016

Bug No known bugs.

This header contains declarations of the key bank tests and the end-to-end gateway tests. These tests are not exhaustive, they test basic functionality of both structures.

3.34 hashTest.cpp File Reference

Implementation for hash tests.

3.34.1 Detailed Description

Implementation for hash tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic hash testing. These tests confirm that the respective hash algorithms are outputting their expected value.

3.35 hashTest.h File Reference

Header for hash testing.

3.35.1 Detailed Description

Header for hash testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic hash algorithms.

3.36 hexConversion.cpp File Reference

Hex conversion implementation.

3.36.1 Detailed Description

Hex conversion implementation.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Implements the set of hex conversion functions. Consult **hexConversion.h** (p. 33) for details.

3.37 hexConversion.h File Reference

Hex conversion header.

Namespaces

- **crypto**

Functions

- bool **crypto::isHexCharacter** (char c)
  Check the character type.
- std::string **crypto::toHex** (unsigned char i)
  Converts an 8 bit integer to a hex string.
- std::string **crypto::toHex** (uint32_t i)
  Converts an 32 bit integer to a hex string.
- unsigned char **crypto::fromHex8** (const std::string &str)
  Converts a hex string to an 8 bit integer.
- uint32_t **crypto::fromHex32** (const std::string &str)
  Converts a hex string to an 32 bit integer.

3.37.1 Detailed Description

Hex conversion header.

Author

Jonathan Bedard

Date

3/16/2016

Bug No known bugs.

Contains a set of functions to convert integers and characters from a hex string and converts hex strings to integers and characters.

3.38 keyBank.cpp File Reference

Implimentation for the AVL tree based key bank.

3.38.1 Detailed Description

Implimentation for the AVL tree based key bank.

Author

Jonathan Bedard

Date

4/19/2016

Bug No known bugs.

This file contians the implimentation for the **crypto::avlKeyBank** (p. 59) and supporting classes. Consult **keyBank.h** (p. 34) for details.

3.39 keyBank.h File Reference

Header for the AVL tree based key bank.

Classes

- class **crypto::nodeGroup**

  Node group.
- class **crypto::nodeNameReference**

  Name storage node.
- class **crypto::nodeKeyReference**

  Key storage node.
- class **crypto::keyBank**

  Key bank interface.
- class **crypto::avlKeyBank**

  AVL key back.

Namespaces

- **crypto**

3.39.1 Detailed Description

Header for the AVL tree based key bank.

Author

Jonathan Bedard

Date

4/19/2016

Bug No known bugs.

This file contians declarations for the **crypto::avlKeyBank** (p. 59) and supporting classes. Note that the key-bank may later be implimented with more advanced datastructures.

3.40 message.cpp File Reference

Crypto-Gateway message implementation.

3.40.1 Detailed Description

Crypto-Gateway message implementation.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

Implements the message used by the crypto-gateway to pass encrypted data between machines.

3.41 message.h File Reference

Crypto-Gateway message.

Classes

- class **crypto::message**
  Crypto-Gateway message.

Namespaces

- **crypto**

3.41.1 Detailed Description

Crypto-Gateway message.

Author

Jonathan Bedard

Date

4/16/2016

Bug No known bugs.

The message declared in this file acts as a message for the Crypto-Gateway. These messages are intended to be converted to machine-to-machine communication.

3.42 publicKeyPackage.cpp File Reference

Implementation of public key bank.

3.42.1 Detailed Description

Implementation of public key bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements a bank of public key types to be accessed at run-time. Essentially acts as a meta-object access bank.

3.43 publicKeyPackage.h File Reference

Declaration of public key bank.

Classes

- class **crypto::publicKeyPackageFrame**
- class **crypto::publicKeyPackage**< **pkType** >
- class **crypto::publicKeyTypeBank**

Namespaces

- **crypto**

3.43.1 Detailed Description

Declaration of public key bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Declares a bank of public keys as well as supporting classes. Acts as a meta-object construct for public-key algorithms.

3.44 publicKeyTest.h File Reference

Public Key tests.

3.44.1 Detailed Description

Public Key tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Since the public key tests are defined by very simple tests, the template testing classes contained in this file are also defined in this file. There is no .cpp file paired with this particular header.

3.45 RC4_Hash.cpp File Reference

3.46 RC4_Hash.h File Reference

Classes

- class **crypto::rc4Hash**
  RC-4 hash class.

Namespaces

- **crypto**

3.47 staticTestKeys.cpp File Reference

Auto-generated.

3.47.1 Detailed Description

Auto-generated.

Author

None

Bug None

3.48 staticTestKeys.h File Reference

Auto-generated.

3.48.1 Detailed Description

Auto-generated.

Author

None

Bug None

3.49 streamCipher.cpp File Reference

3.50 streamCipher.h File Reference

Classes

- class **crypto::streamCipher**
- class **crypto::RCFour**
- class **crypto::streamPacket**
- class **crypto::streamEncrypter**
- class **crypto::streamDecrypter**

Namespaces

- **crypto**

Variables

- bool **global_logging**

3.50.1 Variable Documentation

bool global_logging

3.51 streamPackage.cpp File Reference

Implementation of streaming bank.

3.51.1 Detailed Description

Implementation of streaming bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements a a bank of stream ciphers and hash algorithms to be accessed at run-time. Essentially acts as a meta-object access bank.

3.52 streamPackage.h File Reference

Declaration of streaming bank.

Classes

- class **crypto::streamPackageFrame**
- class **crypto::streamPackage**< **streamType, hashType** >
- class **crypto::streamPackageTypeBank**

Namespaces

- **crypto**

3.52.1 Detailed Description

Declaration of streaming bank.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Declares a bank of stream ciphers and hash algorithms along with supporting classes. Acts as a meta-object construct for public-key algorithms.

3.53 streamTest.cpp File Reference

Implementation for stream tests.

3.53.1 Detailed Description

Implementation for stream tests.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains algorithm-specific cryptographic stream testing. These tests confirm that the respective stream algorithms are outputting their expected value.

3.54 streamTest.h File Reference

Header for stream testing.

3.54.1 Detailed Description

Header for stream testing.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a number of template classes used to confirm the functionality of cryptographic stream objects.

3.55 testKeyGeneration.cpp File Reference

3.56 testKeyGeneration.h File Reference

Implementation of test key binding.

3.56.1 Detailed Description

Implementation of test key binding.
Binds generated testing keys.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the binding of the static test keys to arrays in memory. Consult **testKeyGeneration.h** (p. 41) for details.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

Provides access to the keys generated and stored in **staticTestKeys.h** (p. 38) and **staticTest↩ Keys.cpp** (p. 38). These keys are always copied into a raw array of uint32_t.

3.57 user.cpp File Reference

Implementation of the CryptoGateway user.

3.57.1 Detailed Description

Implementation of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/26/2016

Bug None

Provides an implementation of user which has a user-name, password and associated bank of public keys. Consult **user.h** (p. 42) for details.

3.58 user.h File Reference

Definition of the CryptoGateway user.

Classes

- class **crypto::user**
  Primary user class.

Namespaces

- **crypto**

3.58.1 Detailed Description

Definition of the CryptoGateway user.

Author

Jonathan Bedard

Date

4/26/2016

Bug None

Provides a definition of user which has a user-name, password and associated bank of public keys.

3.59 XMLEncryption.cpp File Reference

Implementation of RC-4.

3.59.1 Detailed Description

Implementation of RC-4.
Implements encrypted XML functions.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements the RC-4 stream cipher and more generally, a framework for all stream ciphers to use.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Implements functions to save and load XML trees in files locked with both a password and with public keys.

3.60 XMLEncryption.h File Reference

Defines basic stream ciphers.

Namespaces

- **crypto**

Functions

- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, unsigned char ∗symKey, unsigned int passwordLength, os::smart_ptr< streamPackageFrame > spf=NULL)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< streamPackageFrame > spf=NULL)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< publicKey > pbk, unsigned int lockType=file::PRIVATE_UNLOCK, os::smart_ptr< streamPackageFrame > spf=NULL)
- bool **crypto::EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< number > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< streamPackageFrame > spf=NULL)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, unsigned char ∗symKey, unsigned int passwordLength)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< publicKey > pbk)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank)
- os::smartXMLNode **crypto::EXML_Input** (std::string path, os::smart_ptr< keyBank > kyBank, os::smart_ptr< nodeGroup > &author)

3.60.1 Detailed Description

Defines basic stream ciphers.

Provides structure to encrypt an XML save file.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Defines some basic stream ciphers and stream cipher tools for basic encryption.

Author

Jonathan Bedard

Date

5/19/2016

Bug None

Provides functions to save and load XML trees in encrypted files.

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 crypto Namespace Reference

Classes

- class **actionOnFileClosed**

  File closed error.
- class **actionOnFileError**

  File error.
- class **avlKeyBank**

  AVL key back.
- class **binaryDecryptor**

  Encrypted binary file output.
- class **binaryEncryptor**

  Encrypted binary file output.
- class **bufferLargeError**

  Buffer too large.
- class **bufferSmallError**

  Buffer too small.
- class **customError**

  Custom ***crypto::error*** *(p. 81).*
- class **error**

  Sortable exception.
- class **errorListener**

  crypto::error *(p. 81) listener*
- class **errorSender**

  Sends ***crypto::error*** *(p. 81).*
- class **fileFormatError**

  File format error.
- class **fileOpenError**

  File open error.

- class **gateway**

  Security gateway.
- class **gatewaySettings**

  Holds settings for gateway encryption.
- class **hash**

  Base hash class.
- class **hashCompareError**

  Hash mis-match.
- class **hashGenerationError**

  Hash generation error.
- class **illegalAlgorithmBind**

  Algorithm bound failure.
- class **insertionFailed**

  ADS Insertion Failed.
- class **integer**

  Integer number definition.
- class **keyBank**

  Key bank interface.
- class **keyChangeReceiver**

  Interface for receiving key changes.
- class **keyChangeSender**

  *Interface inherited by **publicKey** (p. 199).*
- class **keyMissing**

  Key missing error.
- class **masterMismatch**

  Master mis-match.
- class **message**

  Crypto-Gateway message.
- class **nodeGroup**

  Node group.
- class **nodeKeyReference**

  Key storage node.
- class **nodeNameReference**

  Name storage node.
- class **NULLDataError**

  NULL data error.
- class **NULLMaster**

  NULL master error.
- class **NULLPublicKey**

  NULL public-key error.
- class **number**

  Basic number definition.
- class **passwordLargeError**

Symmetric key too big.

- class **passwordSmallError**

  Symmetric key too small.
- class **publicKey**

  Base public-key class.
- class **publicKeyPackage**
- class **publicKeyPackageFrame**
- class **publicKeySizeWrong**

  Public-key size error.
- class **publicKeyTypeBank**
- class **publicRSA**

  RSA public-key encryption.
- class **rc4Hash**

  RC-4 hash class.
- class **RCFour**
- class **RSAKeyGenerator**

  Helper key generation class.
- class **streamCipher**
- class **streamDecrypter**
- class **streamEncrypter**
- class **streamPackage**
- class **streamPackageFrame**
- class **streamPackageTypeBank**
- class **streamPacket**
- class **stringTooLarge**

  String size error.
- class **unknownErrorType**

  Unknown error.
- class **user**

  Primary user class.
- class **xorHash**

  XOR hash class.

Typedefs

- typedef os::smart_ptr< **error** > **errorPointer**

  Smart pointer to ***crypto::error*** (p. *81*).

Functions

- std::ostream & **operator**<< (std::ostream &os, const **hash** &num)

  Output stream operator.
- std::istream & **operator**>> (std::istream &is, **hash** &num)

  Input stream operator.
- template<class hashClass >
  hashClass **hashData** (uint16_t hashType, const unsigned char *data, uint32_t length)

  Hashes data with the specified algorithm.
- std::ostream & **cryptoout_func** ()

  Standard out object for crypto namespace.
- std::ostream & **cryptoerr_func** ()

  Standard error object for crypto namespace.
- std::ostream & **operator**<< (std::ostream &os, const **number** &num)

  Output stream operator.
- std::istream & **operator**>> (std::istream &is, **number** &num)

  Input stream operator.
- bool **isHexCharacter** (char c)

  Check the character type.
- std::string **toHex** (unsigned char i)

  Converts an 8 bit integer to a hex string.
- std::string **toHex** (uint32_t i)

  Converts an 32 bit integer to a hex string.
- unsigned char **fromHex8** (const std::string &str)

  Converts a hex string to an 8 bit integer.
- uint32_t **fromHex32** (const std::string &str)

  Converts a hex string to an 32 bit integer.
- bool **EXML_Output** (std::string path, os::smartXMLNode head, unsigned char *symKey, unsigned int passwordLength, os::smart_ptr< **streamPackageFrame** > spf=NULL)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, std::string password, os::smart↩_ptr< **streamPackageFrame** > spf=NULL)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< **publicKey** > pbk, unsigned int lockType=file::PRIVATE_UNLOCK, os::smart_ptr< **streamPackageFrame** > spf=NULL)
- bool **EXML_Output** (std::string path, os::smartXMLNode head, os::smart_ptr< **number** > **public↩Key**, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > spf=N↩ULL)
- os::smartXMLNode **EXML_Input** (std::string path, unsigned char *symKey, unsigned int password↩Length)
- os::smartXMLNode **EXML_Input** (std::string path, std::string password)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **publicKey** > pbk, os::smart↩_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > &author)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **publicKey** > pbk)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **keyBank** > kyBank)
- os::smartXMLNode **EXML_Input** (std::string path, os::smart_ptr< **keyBank** > kyBank, os↩::smart_ptr< **nodeGroup** > &author)

Variables

- bool **global_logging**

  Deprecated logging flag.
- os::smart_ptr< std::ostream > **cryptoout_ptr**

  Standard out pointer for crypto namespace.
- os::smart_ptr< std::ostream > **cryptoerr_ptr**

  Standard error pointer for crypto namespace.

5.1.1 Typedef Documentation

typedef os::smart_ptr<**error**> **crypto::errorPointer**

Smart pointer to **crypto::error** (p. 81).

5.1.2 Function Documentation

std::ostream& crypto::cryptoerr_func ( )

Standard error object for crypto namespace.
#define statements allow the user to call this function with "crypto::cryptoerr." Logging is achieved by using "crypto::cryptoerr" as one would use "std::cerr."

std::ostream& crypto::cryptoout_func ( )

Standard out object for crypto namespace.
#define statements allow the user to call this function with "crypto::cryptoout." Logging is achieved by using "crypto::cryptoout" as one would use "std::cout."

os::smartXMLNode crypto::EXML_Input (std::string path, unsigned char * symKey, unsigned int passwordLength)

os::smartXMLNode crypto::EXML_Input (std::string path, std::string password)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **publicKey** > pbk, os::smart_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > & author)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **publicKey** > pbk)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **keyBank** > kyBank)

os::smartXMLNode crypto::EXML_Input (std::string path, os::smart_ptr< **keyBank** > kyBank, os::smart_ptr< **nodeGroup** > & author)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, unsigned char * symKey, unsigned int passwordLength, os::smart_ptr< **streamPackageFrame** > spf = `NULL`)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, std::string password, os::smart_ptr< **streamPackageFrame** > spf = `NULL`)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, os::smart_ptr< **publicKey** > pbk, unsigned int lockType = `file::PRIVATE_UNLOCK`, os::smart_ptr< **streamPackageFrame** > spf = `NULL`)

bool crypto::EXML_Output (std::string path, os::smartXMLNode head, os::smart_ptr< **number** > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > spf = `NULL`)

uint32_t crypto::fromHex32 (const std::string & str)

Converts a hex string to an 32 bit integer.

Parameters

`in`	*str*	Hex string to convert

Returns

str converted to integer

unsigned char crypto::fromHex8 (const std::string & str)

Converts a hex string to an 8 bit integer.

Parameters

`in`	*str*	Hex string to convert

Returns

str converted to integer

template<class hashClass > hashClass crypto::hashData (uint16_t hashType, const unsigned char * data, uint32_t length)

Hashes data with the specified algorithm.

Hashes the provided data array returning a hash of the specified algorithm. This is a template function, which calls the static hash function for the specified algorithm.

Parameters

`in`	*hashType*	Size of hash
`in`	*data*	Data array to be hashed
`in`	*length*	Length of data to be hashed

Returns

Hash for data array

bool crypto::isHexCharacter (char c)

Check the character type.

Checks if the character is a valid hex character. That is, 0-9 and A-F.

Parameters

in	*c*	Character to test

Returns

true if a hex character, else, false

std::ostream& crypto::operator<< (std::ostream & os, const **number** & num)

Output stream operator.

Parameters

	[in/out]	os Output stream
in	*num*	Number to be output

Returns

reference to std::ostream& os

std::ostream& crypto::operator<< (std::ostream & os, const **hash** & num)

Output stream operator.

Outputs a hex version of the hash to the provided output stream. This output will look identical for two hashes which are equal but have different algorithms.

Parameters

	[in/out]	os Output stream
in	*num*	Hash to be printed return Reference to output stream

std::istream& crypto::operator>> (std::istream & is, **number** & num)

Input stream operator.

Parameters

	[in/out]	is Input stream
in	*num*	Number to set with the string

Returns

reference to std::istream& is

std::istream& crypto::operator>> (std::istream & is, **hash** & num)

Input stream operator.

Inputs a hex version of the hash from the provided output stream. This function must receive a constructed hash, although it will rebuild the provided hash with the stream data.

Parameters

	[in/out]	is Input stream
`in`	*num*	Hash to be created return Reference to input stream

std::string crypto::toHex (unsigned char i)

Converts an 8 bit integer to a hex string.

Parameters

`in`	*i*	Integer to convert

Returns

i converted to hex string

std::string crypto::toHex (uint32_t i)

Converts an 32 bit integer to a hex string.

Parameters

`in`	*i*	Integer to convert

Returns

i converted to hex string

5.1.3 Variable Documentation

os::smart_ptr<std::ostream> crypto::cryptoerr_ptr

Standard error pointer for crypto namespace.

This std::ostream is used as standard error for the crypto namespace. This pointer can be swapped out to programmatically redirect standard error for the crypto namespace.

os::smart_ptr<std::ostream> crypto::cryptoout_ptr

Standard out pointer for crypto namespace.

This std::ostream is used as standard out for the crypto namespace. This pointer can be swapped out to programmatically redirect standard out for the crypto namespace.

bool crypto::global_logging

Deprecated logging flag.

Old logging flag. Deprecated in the new CryptoGateway files. This has been replaced by the logging system outlined in this file.

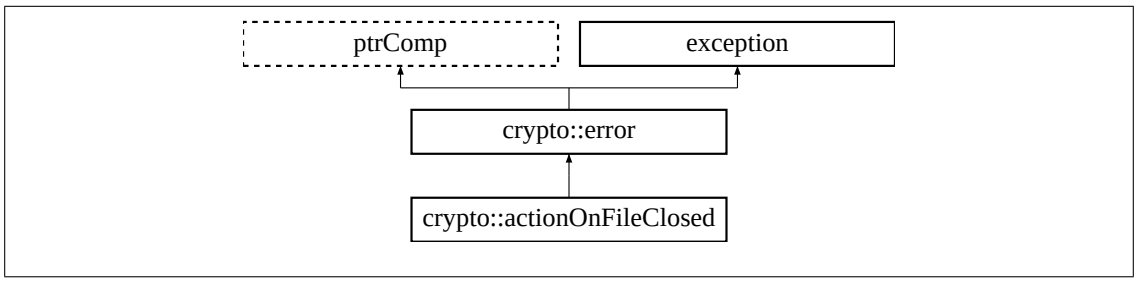
Chapter 6

Class Documentation

6.1 crypto::actionOnFileClosed Class Reference

File closed error.

Inheritance diagram for crypto::actionOnFileClosed:



Public Member Functions

- virtual ~**actionOnFileClosed** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Action on File Closed".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot preform action on a file in the closed state".

6.1.1 Detailed Description

File closed error.

Thrown when an action is attempted on a file which is already closed.

6.1.2 Constructor & Destructor Documentation

virtual crypto::actionOnFileClosed::~actionOnFileClosed ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.1.3 Member Function Documentation

std::string crypto::actionOnFileClosed::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot preform action on a file in the closed state".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::actionOnFileClosed::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Action on File Closed".

Returns

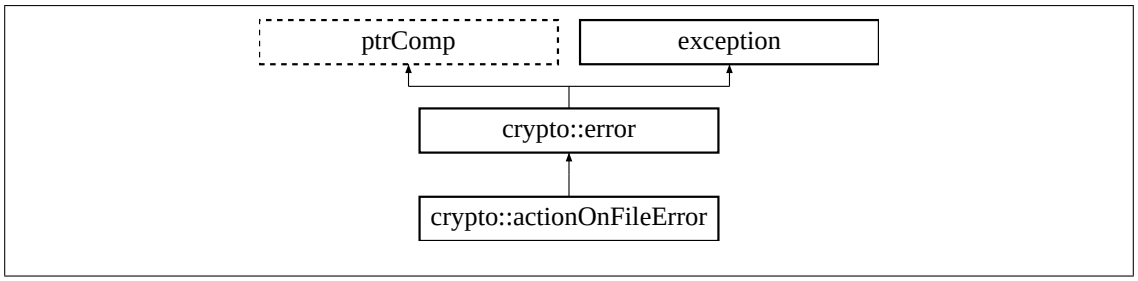
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.2 crypto::actionOnFileError Class Reference

File error.

Inheritance diagram for crypto::actionOnFileError:



Public Member Functions

- virtual ~**actionOnFileError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Action on File Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot preform action on a file in the error state".

6.2.1 Detailed Description

File error.

Thrown when an action is attempted on a file in the error state.

6.2.2 Constructor & Destructor Documentation

virtual crypto::actionOnFileError::~actionOnFileError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.2.3 Member Function Documentation

std::string crypto::actionOnFileError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot preform action on a file in the error state".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::actionOnFileError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Action on File Error".

Returns

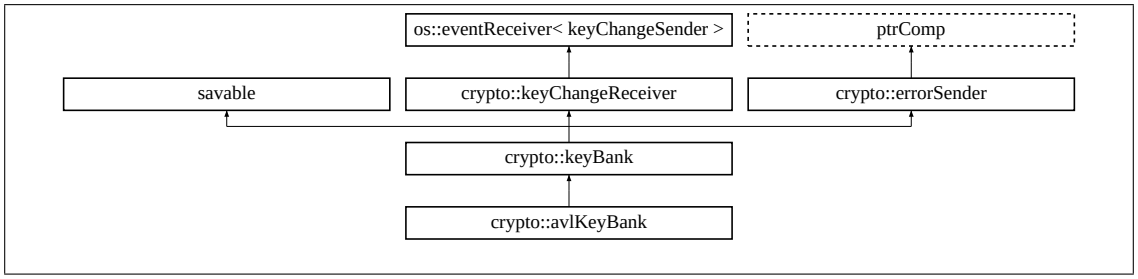
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.3 crypto::avlKeyBank Class Reference

AVL key back.

Inheritance diagram for crypto::avlKeyBank:



Public Member Functions

- **avlKeyBank** (std::string **savePath**="", const unsigned char *key=NULL, unsigned int keyLen=0, os::smart_ptr< **streamPackageFrame** > strmPck=NULL)

  Construct with save path.
- **avlKeyBank** (std::string **savePath**, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **streamPackageFrame** > strmPck=NULL)

  Construct with save path and public key.
- virtual ~**avlKeyBank** ()

  Virtual destructor.
- void **save** ()

  Saves bank to file.
- os::smart_ptr< **nodeGroup** > **addPair** (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Adds authenticated node to bank.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeNameReference** > name)

  Find by group name reference.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeKeyReference** > key)

  Find by group key reference.
- os::smart_ptr< **nodeGroup** > **find** (std::string groupName, std::string name)

  Find by group name and name.
- os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Find by key information.

Protected Member Functions

- void **pushNewNode** (os::smart_ptr< **nodeNameReference** > name)

  Add name node.
- void **pushNewNode** (os::smart_ptr< **nodeKeyReference** > key)

  Add key node.
- void **load** ()

  Loads bank from file.

Private Attributes

- os::asyncAVLTree< **nodeNameReference** > **nameTree**

  List of all names associated with this node.
- os::asyncAVLTree< **nodeKeyReference** > **keyTree**

  List of all keys associated with this node.
- os::asyncAVLTree< **nodeGroup** > **nodeBank**

  List of all node groups.

Additional Inherited Members

6.3.1 Detailed Description

AVL key back.

The AVL key bank stores keys in a series of AVL trees. All keys in the bank are loaded into memory when the file is loaded, meaning that there is a limited number of keys that can be practically managed through an AVL key bank.

6.3.2 Constructor & Destructor Documentation

crypto::avlKeyBank::avlKeyBank (std::string savePath = "", const unsigned char * key = NULL, unsigned int keyLen = 0, os::smart_ptr< **streamPackageFrame** > strmPck = NULL)

Construct with save path.

Intializes the key bank and loads the the bank from a file.

Parameters

in	*savePath*	Path to save file, empty by default
in	*key*	Symetric key
in	*keyLen*	Length of symetric key
in	*strmPck*	Definition of algorithms used

crypto::avlKeyBank::avlKeyBank (std::string savePath, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **streamPackageFrame** > strmPck = NULL)

Construct with save path and public key.

Intializes the key bank and loads the the bank from a file.

Parameters

in	*savePath*	Path to save file
in	*pubKey*	Public key
in	*strmPck*	Definition of algorithms used

virtual crypto::avlKeyBank::~avlKeyBank ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.3.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::addPair (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) `[virtual]`

Adds authenticated node to bank.

Note that if a node has not be authenticated, adding it to the bank will cause a potential security vulnerability. Nodes should be authenticated before being added to the bank.

Parameters

`in`	*groupName*	Name of the node's group
`in`	*name*	Name of the node
`in`	*key*	Key of node to be added
`in`	*algoID*	ID of algorithm for key
`in`	*keySize*	Length of key of the node

Returns

Return reference to the new node group

Implements **crypto::keyBank** (p. 139).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **nodeNameReference** > name) `[virtual]`

Find by group name reference.

Parameters

`in`	*name*	Name reference to be searched

Returns

Node group found by arguments

Implements **crypto::keyBank** (p. 140).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **nodeKeyReference** > key) `[virtual]`

Find by group key reference.

Parameters

`in`	*key*	Key reference to be searched

Returns

Node group found by arguments

Implements **crypto::keyBank** (p. 140).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (std::string groupName, std::string name) `[inline]`, `[virtual]`

Find by group name and name.

Parameters

`in`	*groupName*	Name of the node's group
`in`	*name*	Name of the node

Returns

Node group found by arguments

Reimplemented from **crypto::keyBank** (p. 140).

os::smart_ptr<**nodeGroup**> crypto::avlKeyBank::find (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) `[inline]`, `[virtual]`

Find by key information.

Parameters

`in`	*key*	Key of node to be added
`in`	*algoID*	ID of algorithm for key
`in`	*keySize*	Length of key of the node

Returns

Node group found by arguments

Reimplemented from **crypto::keyBank** (p. 141).

void crypto::avlKeyBank::load ( ) `[protected]`, `[virtual]`

Loads bank from file.

Returns

void

Implements **crypto::keyBank** (p. 141).

void crypto::avlKeyBank::pushNewNode (os::smart_ptr< **nodeNameReference** > name) [protected], [virtual]

Add name node.

Inserts a name node into the bank. The name node has a reference to a node group.

Parameters

in	*name*	Name node to be added

Returns

void

Implements **crypto::keyBank** (p. 141).

void crypto::avlKeyBank::pushNewNode (os::smart_ptr< **nodeKeyReference** > key) [protected], [virtual]

Add key node.

Inserts a key node into the bank. The key node has a reference to a node group.

Parameters

in	*key*	Key node to be added

Returns

void

Implements **crypto::keyBank** (p. 142).

void crypto::avlKeyBank::save ( ) [virtual]

Saves bank to file.

Returns

void

Implements **crypto::keyBank** (p. 142).

6.3.4 Member Data Documentation

os::asyncAVLTree<**nodeKeyReference**> crypto::avlKeyBank::keyTree [private]

List of all keys associated with this node.

os::asyncAVLTree<**nodeNameReference**> crypto::avlKeyBank::nameTree [private]

List of all names associated with this node.

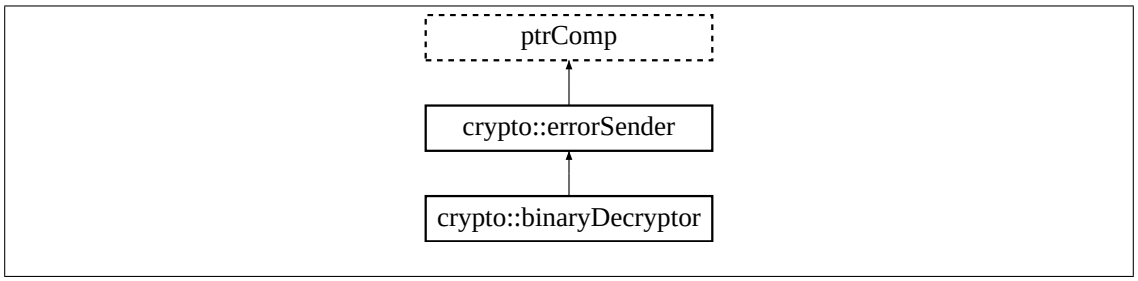
os::asyncAVLTree<**nodeGroup**> crypto::avlKeyBank::nodeBank `[private]`

List of all node groups.

6.4 crypto::binaryDecryptor Class Reference

Encrypted binary file output.

Inheritance diagram for crypto::binaryDecryptor:



Public Member Functions

- **binaryDecryptor** (std::string file_name, os::smart_ptr< **keyBank** > kBank)
  Construct with public key.
- **binaryDecryptor** (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock)
  Construct with public key.
- **binaryDecryptor** (std::string file_name, std::string password)
  Construct with password.
- **binaryDecryptor** (std::string file_name, unsigned char *key, unsigned int keyLen)
  Construct with symmetric key.
- unsigned char **read** ()
  Attempts to read a single character.
- unsigned int **read** (unsigned char *data, unsigned int dataLen)
  Attempts to read a block of data.
- void **close** ()
  Closes the output file.
- const std::string & **fileName** () const
  Returns the name of target file.
- const os::smart_ptr< **streamPackageFrame** > **streamAlgorithm** () const
  Returns the stream algorithm definition.
- bool **good** () const
  Returns the current file state.
- bool **finished** () const
  Returns if the file has finished writing.
- unsigned long **bytesLeft** () const
  Returns the number of bytes left in the file.

- os::smart_ptr< **nodeGroup** > **author** ()

  Pointer to the user which signed this file.
- virtual ~**binaryDecryptor** ()

  Virtual destructor.

Private Member Functions

- void **build** (unsigned char *key=NULL, unsigned int keyLen=0)

  Central constructor function.

Private Attributes

- os::smart_ptr< **publicKey** > **_publicKeyLock**

  Pointer to the optional public key.
- os::smart_ptr< **keyBank** > **_keyBank**

  Pointer to the key bank (to confirm public keys)
- os::smart_ptr< **nodeGroup** > **_author**

  Pointer to the user which signed this file.
- os::smart_ptr< **streamPackageFrame** > **_streamAlgorithm**

  Pointer to the mandatory stream algorithm definition.
- os::smart_ptr< **streamCipher** > **currentCipher**

  Pointer to the current stream cipher.
- bool **_state**

  State of the output file.
- bool **_finished**

  Has the file been closed.
- std::string **_fileName**

  Name of the file being read from.
- std::ifstream **input**

  Binary input file.
- unsigned long **_bytesLeft**

  Number of bytes left in the file.

Additional Inherited Members

6.4.1 Detailed Description

Encrypted binary file output.

The user defines an encryption algorithm and key, then places data into the file. This data is automatically encrypted with the specified algorithm and key.

6.4.2 Constructor & Destructor Documentation

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, os::smart_ptr< **keyBank** > kBank)

Construct with public key.

Constructs the file reader with a public key.

Parameters

in	*file_name*	Name of input file
in	*kBank*	Record of public keys

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock)

Construct with public key.

Constructs the file reader with a public key.

Parameters

in	*file_name*	Name of input file
in	*publicKeyLock*	Public key to decrypt data

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, std::string password)

Construct with password.

Constructs the file reader with a password.

Parameters

in	*file_name*	Name of input file
in	*password*	Password to decrypt data

crypto::binaryDecryptor::binaryDecryptor (std::string file_name, unsigned char * key, unsigned int keyLen)

Construct with symmetric key.

Constructs the file reader with a symmetric key.

Parameters

in	*file_name*	Name of input file
in	*key*	Symmetric key byte array
in	*keyLen*	Size of the symmetric key

virtual crypto::binaryDecryptor::~binaryDecryptor ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Also closes the input file.

6.4.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::binaryDecryptor::author ( )

Pointer to the user which signed this file.

Returns

crypto::binaryDecryptor::_author (p. 70)

void crypto::binaryDecryptor::build (unsigned char * key = NULL, unsigned int keyLen = 0) [private]

Central constructor function.

This function reads the header of the encrypted binary file and attempts to initialize a stream cipher for decryption. Note that there is no guarantee that this can be done with the information given to the class. In this event, the class logs the error and sets it's state to false.

Parameters

in	*key*	Symmetric key, NULL by default
in	*keyLen*	Length of symmetric key, 0 by default

Returns

void

unsigned long crypto::binaryDecryptor::bytesLeft ( ) const [inline]

Returns the number of bytes left in the file.

Returns

crypto::binaryDecryptor::_bytesLeft (p. 70)

void crypto::binaryDecryptor::close ( )

Closes the output file.

Returns

void

const std::string& crypto::binaryDecryptor::fileName ( ) const [inline]

Returns the name of target file.

Returns

crypto::binaryDecryptor::_fileName (p. 70)

bool crypto::binaryDecryptor::finished ( ) const [inline]

Returns if the file has finished writing.

Returns

crypto::binaryDecryptor::_finished (p. 70)

bool crypto::binaryDecryptor::good ( ) const [inline]

Returns the current file state.

Returns

crypto::binaryDecryptor::_state (p. 70)

unsigned char crypto::binaryDecryptor::read ( )

Attempts to read a single character.

Note that if the reader is in a "good" state, then this function will read and decrypt a single byte of the file.

Returns

Character read, 0 if failed

unsigned int crypto::binaryDecryptor::read (unsigned char * data, unsigned int dataLen)

Attempts to read a block of data.

Note that if the reader is in a "good" state, then this function will read and decrypt the entire block of data requested.

Parameters

out	*data*	Array to place read data into
in	*dataLen*	Number of bytes attempting to read

Returns

Number of bytes read

const os::smart_ptr<**streamPackageFrame**> crypto::binaryDecryptor::streamAlgorithm ( ) const [inline]

Returns the stream algorithm definition.

Returns

crypto::binaryDecryptor::_streamAlgorithm (p. 70)

6.4.4 Member Data Documentation

os::smart_ptr<**nodeGroup**> crypto::binaryDecryptor::_author [private]

Pointer to the user which signed this file.
This is only populated if a key-bank is bound to the class.

unsigned long crypto::binaryDecryptor::_bytesLeft [private]

Number of bytes left in the file.

std::string crypto::binaryDecryptor::_fileName [private]

Name of the file being read from.

bool crypto::binaryDecryptor::_finished [private]

Has the file been closed.
If true, the file is closed. Else, the file is open and may be read from.

os::smart_ptr<**keyBank**> crypto::binaryDecryptor::_keyBank [private]

Pointer to the key bank (to confirm public keys)

os::smart_ptr<**publicKey**> crypto::binaryDecryptor::_publicKeyLock [private]

Pointer to the optional public key.

bool crypto::binaryDecryptor::_state [private]

State of the output file.
This state is either "good" or "bad." A bad file is not merely defined by crypto::binaryEncryptor↩
::input, but also by any cryptographic abnormalities that are detected.

os::smart_ptr<**streamPackageFrame**> crypto::binaryDecryptor::_streamAlgorithm [private]

Pointer to the mandatory stream algorithm definition.

os::smart_ptr<**streamCipher**> crypto::binaryDecryptor::currentCipher [private]

Pointer to the current stream cipher.
The current cipher will be of the type defined in the algorithm definition. It will be initialized with either the provided public key or the provided password.

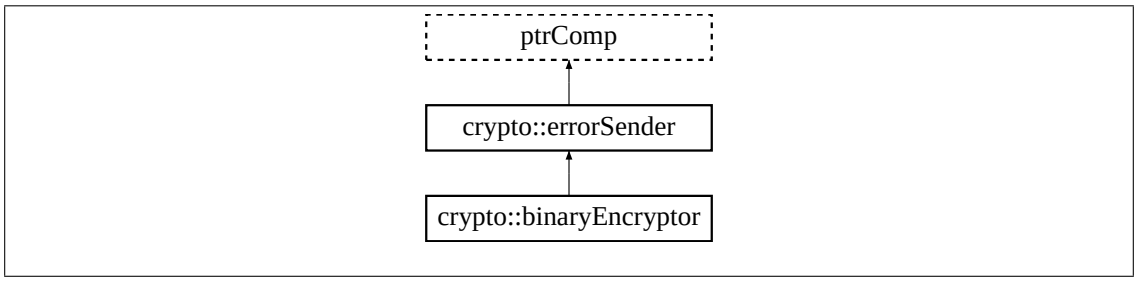
std::ifstream crypto::binaryDecryptor::input [private]

Binary input file.

6.5 crypto::binaryEncryptor Class Reference

Encrypted binary file output.

Inheritance diagram for crypto::binaryEncryptor:



Public Member Functions

- **binaryEncryptor** (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock, unsigned int lockType=file::PRIVATE_UNLOCK, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with public key.
- **binaryEncryptor** (std::string file_name, os::smart_ptr< **number** > **publicKey**, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with number and public key algorithm.
- **binaryEncryptor** (std::string file_name, std::string password, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with password.
- **binaryEncryptor** (std::string file_name, unsigned char *key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with symmetric key.
- void **write** (unsigned char data)

  Write a single character.
- void **write** (const unsigned char *data, unsigned int dataLen)

  Write an array of bytes.
- void **close** ()

  Closes the output file.
- const std::string & **fileName** () const

  Returns the name of target file.
- const os::smart_ptr< **streamPackageFrame** > **streamAlgorithm** () const

  Returns the stream algorithm definition.
- bool **good** () const

  Returns the current file state.
- bool **finished** () const

  Returns if the file has finished writing.
- virtual ~**binaryEncryptor** ()

  Virtual destructor.

Private Member Functions

- void **build** (unsigned char *key, unsigned int keyLen)

  Construct class with password.
- void **build** (os::smart_ptr< **publicKey** > publicKeyLock)

  Construct class with public key.
- void **build** (os::smart_ptr< **number** > pubKey, unsigned int pkAlgo, unsigned int pkSize)

  Construct class with number and algorithm.

Private Attributes

- unsigned int **_publicLockType**

  Defines method of locking the file.
- os::smart_ptr< **streamPackageFrame** > **_streamAlgorithm**

  Pointer to the mandatory stream algorithm definition.
- os::smart_ptr< **streamCipher** > **currentCipher**

  Pointer to the current stream cipher.
- bool **_state**

  State of the output file.
- bool **_finished**

  Has the file been closed.
- std::string **_fileName**

  Name of the file being written to.
- std::ofstream **output**

  Binary output file.

Additional Inherited Members

6.5.1 Detailed Description

Encrypted binary file output.

The user defines an encryption algorithm and key, then places data into the file. This data is automatically encrypted with the specified algorithm and key.

6.5.2 Constructor & Destructor Documentation

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, os::smart_ptr< **publicKey** > publicKeyLock, unsigned int lockType = `file::PRIVATE_UNLOCK`, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with public key.

Constructs the file writer with a public key and an optional stream algorithm definition

Parameters

`in`	*file_name*	Name of output file
`in`	*publicKeyLock*	Public key to encrypt data

Parameters

in	*lockType*	Defines method of locking with public key
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, os::smart_ptr< **number** > publicKey, unsigned int pkAlgo, unsigned int pkSize, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with number and public key algorithm.

Constructs the file writer with a public key and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	***publicKey*** *(p. 199)*	Number to encrypt data
in	*pkAlgo*	Defines public key algorithm
in	*pkSize*	Defines size of public key
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, std::string password, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with password.

Constructs the file writer with a password and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	*password*	String to encrypt data with
in	*stream_algo*	Optional stream algorithm definition

crypto::binaryEncryptor::binaryEncryptor (std::string file_name, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = NULL)

Construct with symmetric key.

Constructs the file writer with a symmetric key and an optional stream algorithm definition

Parameters

in	*file_name*	Name of output file
in	*key*	Array of characters defining the symmetric key
in	*keyLen*	Length of symmetric key

Parameters

in	*stream_algo*	Optional stream algorithm definition

virtual crypto::binaryEncryptor::~binaryEncryptor ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Also closes the output file.

6.5.3 Member Function Documentation

void crypto::binaryEncryptor::build (unsigned char * key, unsigned int keyLen) [private]

Construct class with password.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple data formats to be converted into the key.

Parameters

in	*key*	Array of characters defining the symmetric key
in	*keyLen*	Length of symmetric key

Returns

void

void crypto::binaryEncryptor::build (os::smart_ptr< **publicKey** > publicKeyLock) [private]

Construct class with public key.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple types of data to be converted to a public key.

Parameters

in	*publicKeyLock*	Public key pair to encrypt data

Returns

void

void crypto::binaryEncryptor::build (os::smart_ptr< **number** > pubKey, unsigned int pkAlgo, unsigned int pkSize) [private]

Construct class with number and algorithm.

This function acts as a constructor. It is only called by "true" constructors and exists to allow multiple types of data to be converted to a public key.

Parameters

in	*pubKey*	Public key to encrypt data
in	*pkAlgo*	Algorithm ID
in	*pkSize*	Size of public key

Returns

void

void crypto::binaryEncryptor::close ( )

Closes the output file.

Returns

void

const std::string& crypto::binaryEncryptor::fileName ( ) const [inline]

Returns the name of target file.

Returns

crypto::binaryEncryptor::_fileName (p. 76)

bool crypto::binaryEncryptor::finished ( ) const [inline]

Returns if the file has finished writing.

Returns

crypto::binaryEncryptor::_finished (p. 76)

bool crypto::binaryEncryptor::good ( ) const [inline]

Returns the current file state.

Returns

crypto::binaryEncryptor::_state (p. 76)

const os::smart_ptr<**streamPackageFrame**> crypto::binaryEncryptor::streamAlgorithm ( ) const [inline]

Returns the stream algorithm definition.

Returns

crypto::binaryEncryptor::_streamAlgorithm (p. 76)

void crypto::binaryEncryptor::write (unsigned char data)

Write a single character.

Parameters

in	*data*	Character to write

Returns

void

void crypto::binaryEncryptor::write (const unsigned char * data, unsigned int dataLen)

Write an array of bytes.

Parameters

in	*data*	Data array to write
in	*dataLen*	Length of data array

Returns

void

6.5.4 Member Data Documentation

std::string crypto::binaryEncryptor::_fileName [private]

Name of the file being written to.

bool crypto::binaryEncryptor::_finished [private]

Has the file been closed.
If true, the file is closed. Else, the file is open and may be written to.

unsigned int crypto::binaryEncryptor::_publicLockType [private]

Defines method of locking the file.

bool crypto::binaryEncryptor::_state [private]

State of the output file.
This state is either "good" or "bad." A bad file is not merely defined by **crypto::binaryEncryptor::output** (p. 77), but also by any cryptographic abnormalities that are detected.

os::smart_ptr<**streamPackageFrame**> crypto::binaryEncryptor::_streamAlgorithm [private]

Pointer to the mandatory stream algorithm definition.

os::smart_ptr<**streamCipher**> crypto::binaryEncryptor::currentCipher [private]

Pointer to the current stream cipher.

The current cipher will be of the type defined in the algorithm definition. It will be initialized with either the provided public key or the provided password.

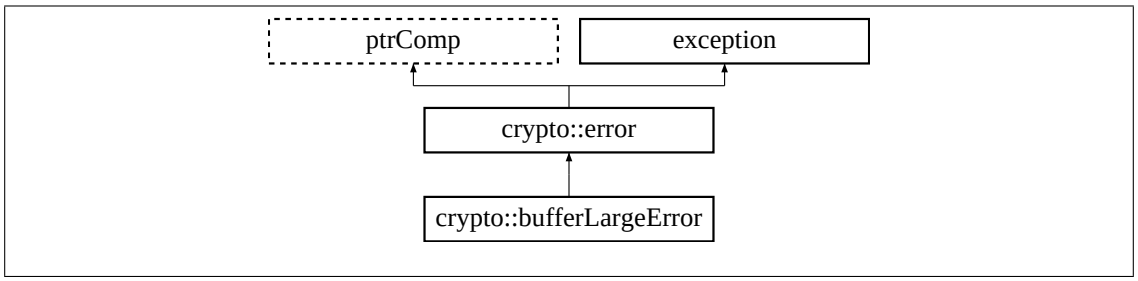
std::ofstream crypto::binaryEncryptor::output [private]

Binary output file.

6.6 crypto::bufferLargeError Class Reference

Buffer too large.

Inheritance diagram for crypto::bufferLargeError:



Public Member Functions

- virtual ~**bufferLargeError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Buffer Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Buffer too large".

6.6.1 Detailed Description

Buffer too large.

Thrown when the buffer provided to some cryptographic function is too large.

6.6.2 Constructor & Destructor Documentation

virtual crypto::bufferLargeError::~bufferLargeError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.6.3 Member Function Documentation

std::string crypto::bufferLargeError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Buffer too large".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::bufferLargeError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Buffer Size Error".

Returns

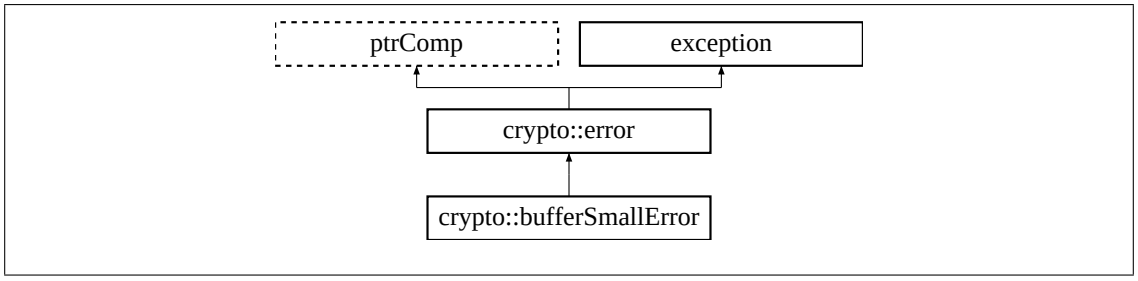
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.7 crypto::bufferSmallError Class Reference

Buffer too small.

Inheritance diagram for crypto::bufferSmallError:



Public Member Functions

- virtual ~**bufferSmallError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Buffer Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Buffer too small".

6.7.1 Detailed Description

Buffer too small.

Thrown when the buffer provided to some cryptographic function is too small.

6.7.2 Constructor & Destructor Documentation

virtual crypto::bufferSmallError::~bufferSmallError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.7.3 Member Function Documentation

std::string crypto::bufferSmallError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Buffer too small".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::bufferSmallError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Buffer Size Error".

Returns

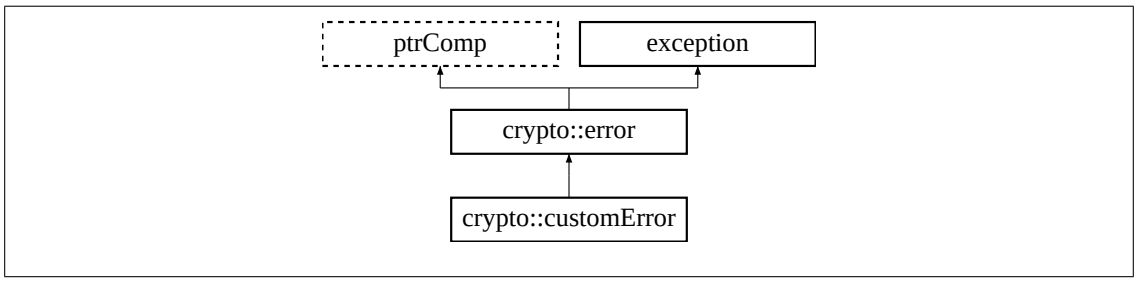
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.8 crypto::customError Class Reference

Custom **crypto::error** (p. 81).

Inheritance diagram for crypto::customError:



Public Member Functions

- **customError** (std::string name, std::string description)

  Custom error constructor.
- virtual ~**customError** () throw ()

  Virtual destructor.

- std::string **errorTitle** () const

  *Short error descriptor Returns "<name>" (**crypto::customError::_name** (p. 81))*
- std::string **errorDescription** () const

  *Long error descriptor Returns "<description>" (**crypto::customError::_description** (p. 81))*

Private Attributes

- std::string **_name**
- std::string **_description**

6.8.1 Detailed Description

Custom **crypto::error** (p. 81).

Allows the programmer to define an error unique to a specific situation.

6.8.2 Constructor & Destructor Documentation

crypto::customError::customError (std::string name, std::string description) `[inline]`

Custom error constructor.

Parameters

`in`	*name*	Short error tag
`in`	*description*	Long error description

virtual crypto::customError::~customError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.8.3 Member Function Documentation

std::string crypto::customError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "<description>" (**crypto::customError::_description** (p. 81))

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::customError::errorTitle ( ) const `[inline]`, `[virtual]`

Short error descriptor Returns "<name>" (**crypto::customError::_name** (p. 81))

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.8.4 Member Data Documentation

std::string crypto::customError::_description [private]

@ Long error descriptor

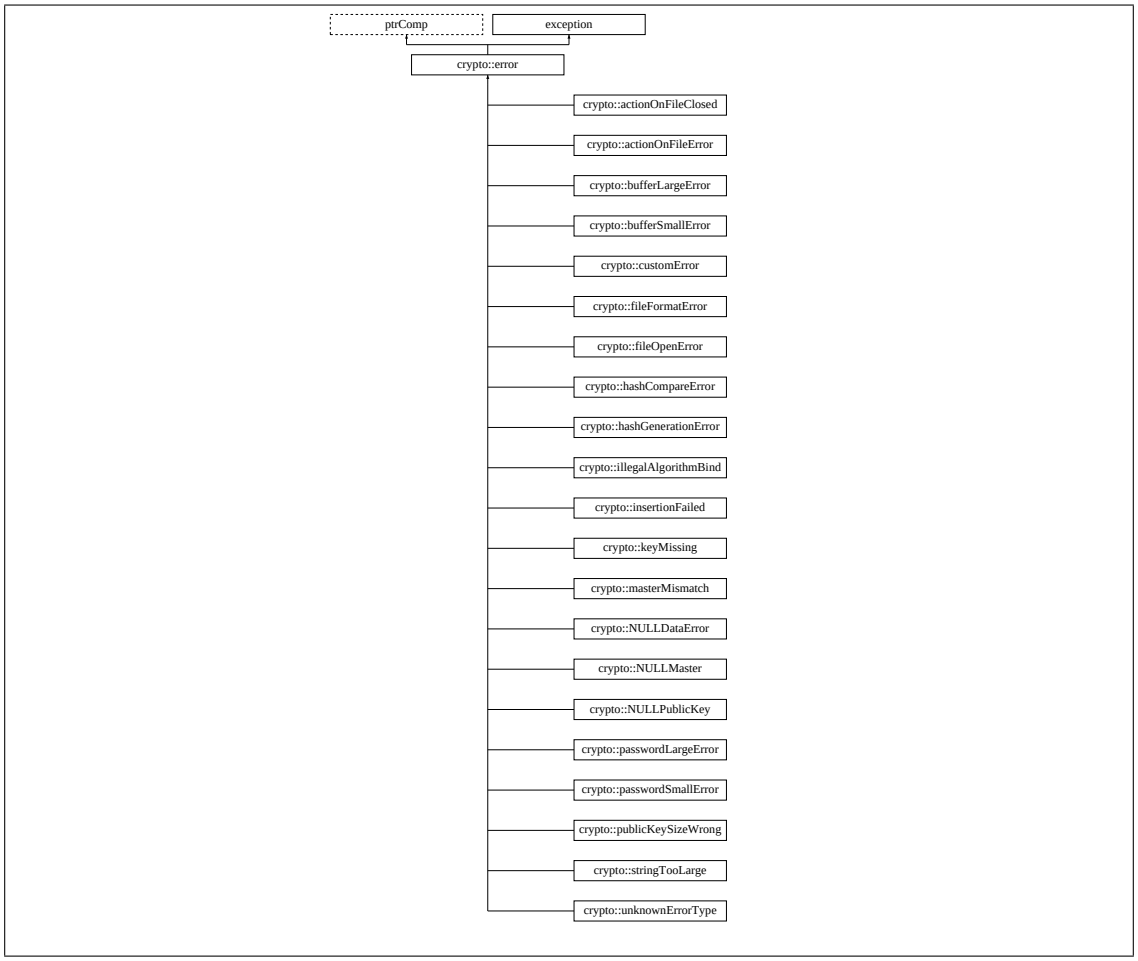
std::string crypto::customError::_name [private]

@ Short error descriptor

6.9 crypto::error Class Reference

Sortable exception.

Inheritance diagram for crypto::error:



Public Member Functions

- **error** ()

  Error constructor.
- virtual ~**error** () throw ()

  Virtual destructor.
- virtual std::string **errorTitle** () const

  Short error descriptor Returns "Error".
- virtual std::string **errorDescription** () const

  Long error descriptor Returns "No description".
- std::string **timestampString** () const

  Timestamp converted to string Returns the timestamp in a human readable string.
- void **log** () const

  Logs error to crypto::cryptoerr Logs the error title, time created and error description on the Crypto↩ Gateway error log.
- uint64_t **timestamp** () const

Time created.

- const char * **what** () const throw ()

  Concatenated error data Returns a C string of the error title, time constructed and error description.

Private Attributes

- uint64_t **_timestamp**

  Time the error was created.

- std::string **whatString**

  Full error output.

6.9.1 Detailed Description

Sortable exception.

This class allows for more sophisticated logging of errors. It contains the time which the error occurred and can be thrown.

6.9.2 Constructor & Destructor Documentation

crypto::error::error ( ) `[inline]`

Error constructor.

Constructs an error by setting the timestamp to the current time.

virtual crypto::error::~error ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.9.3 Member Function Documentation

virtual std::string crypto::error::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "No description".

Returns

Error description std::string

Reimplemented in **crypto::stringTooLarge** (p. 253), **crypto::unknownErrorType** (p. 255), **crypto::masterMismatch** (p. 150), **crypto::NULLMaster** (p. 176), **crypto::NULLDataError** (p. 174), **crypto::NULLPublicKey** (p. 177), **crypto::keyMissing** (p. 149), **crypto::publicKeySizeWrong** (p. 225), **crypto::actionOnFileClosed** (p. 58), **crypto::actionOnFileError** (p. 59), **crypto::hashGenerationError** (p. 124), **crypto::hashCompareError** (p. 122), **crypto::illegalAlgorithmBind** (p. 125), **crypto::fileFormatError** (p. 91), **crypto::fileOpenError** (p. 92), **crypto::customError** (p. 80), **crypto::insertionFailed** (p. 126), **crypto::bufferLargeError** (p. 78), **crypto::bufferSmallError** (p. 79), **crypto::passwordLargeError** (p. 197), and **crypto::passwordSmallError** (p. 198).

virtual std::string crypto::error::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Error".

Returns

Error title std::string

Reimplemented in **crypto::stringTooLarge** (p. 254), **crypto::unknownErrorType** (p. 255), **crypto::masterMismatch** (p. 150), **crypto::NULLMaster** (p. 176), **crypto::NULLDataError** (p. 175), **crypto::NULLPublicKey** (p. 177), **crypto::keyMissing** (p. 149), **crypto::publicKeySizeWrong** (p. 225), **crypto::actionOnFileClosed** (p. 58), **crypto::actionOnFileError** (p. 59), **crypto::hashGenerationError** (p. 124), **crypto::hashCompareError** (p. 123), **crypto::illegalAlgorithmBind** (p. 125), **crypto::fileFormatError** (p. 91), **crypto::fileOpenError** (p. 92), **crypto::customError** (p. 80), **crypto::insertionFailed** (p. 127), **crypto::bufferLargeError** (p. 78), **crypto::bufferSmallError** (p. 79), **crypto::passwordLargeError** (p. 197), and **crypto::passwordSmallError** (p. 199).

void crypto::error::log ( ) const [inline]

Logs error to crypto::cryptoerr Logs the error title, time created and error description on the CryptoGateway error log.

Returns

void

uint64_t crypto::error::timestamp ( ) const [inline]

Time created.

Returns

crypto::error::_timestamp (p. 84)

std::string crypto::error::timestampString ( ) const [inline]

Timestamp converted to string Returns the timestamp in a human readable string.

Returns

Time error was created

const char* crypto::error::what ( ) const throw) [inline]

Concatenated error data Returns a C string of the error title, time constructed and error description.

Returns

Character pointer to error data

6.9.4 Member Data Documentation

uint64_t crypto::error::_timestamp [private]

Time the error was created.

std::string crypto::error::whatString [private]

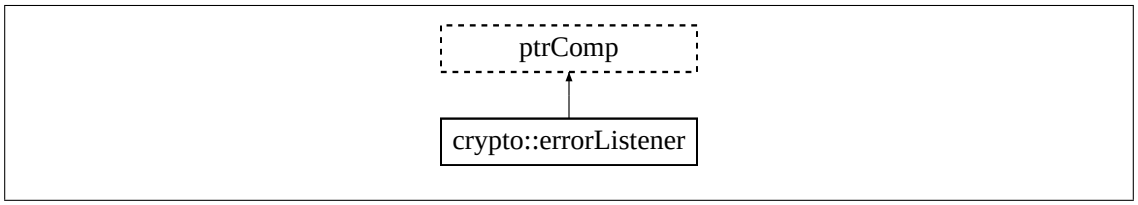
Full error output.

The **crypto::error::what()** (p. 84) function must return a C string. This string is the position in memory that function returns. **crypto::error::what()** (p. 84) also constructs this string.

6.10 crypto::errorListener Class Reference

crypto::error (p. 81) listener

Inheritance diagram for crypto::errorListener:



Public Member Functions

- virtual ~**errorListener** ()

  Virtual destructor.
- virtual void **receiveError** (**errorPointer** elm, os::smart_ptr< **errorSender** > source)

  Receive error event.

Private Attributes

- os::spinLock **mtx**

  Set protection mutex.
- os::smartSet< **errorSender** > **senders**

  Set of senders.

Friends

- class **errorSender**

  *Friendship with **crypto::errorSender** (p. 86).*

6.10.1 Detailed Description

crypto::error (p. 81) listener

Defines a class which is notified when another class throws a **crypto::error** (p. 81).

6.10.2 Constructor & Destructor Documentation

virtual crypto::errorListener::~errorListener ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.10.3 Member Function Documentation

virtual void crypto::errorListener::receiveError (**errorPointer** elm, os::smart_ptr< **errorSender** > source) [inline], [virtual]

Receive error event.

Receives error from one of the senders this listener is registered to.

Parameters

in	*elm*	Error sent
in	*source*	Sender which sent error

Returns

void

6.10.4 Friends And Related Function Documentation

friend class **errorSender** [friend]

Friendship with **crypto::errorSender** (p. 86).

The error sender must be able to add and remove itself from the listener's set.

6.10.5 Member Data Documentation

os::spinLock crypto::errorListener::mtx [private]

Set protection mutex.

Protects access to the set of senders, allows for multi-threading.

os::smartSet<**errorSender**> crypto::errorListener::senders [private]

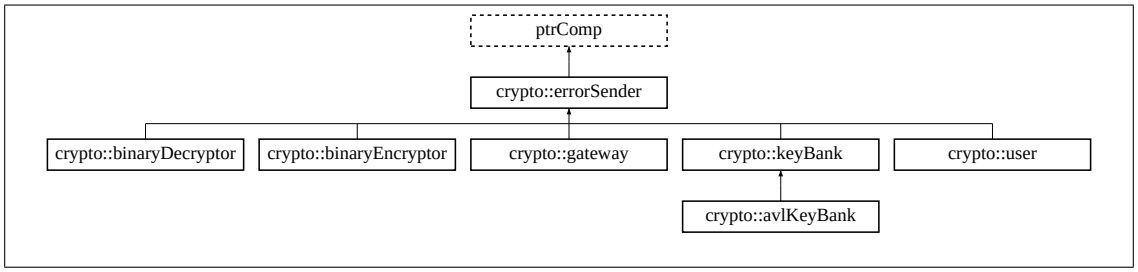
Set of senders.

All of the senders this listener is registered to.

6.11 crypto::errorSender Class Reference

Sends **crypto::error** (p. 81).

Inheritance diagram for crypto::errorSender:



Public Member Functions

- **errorSender** ()

  Error sender constructor.
- virtual ~**errorSender** ()

  Virtual destructor.
- void **pushErrorListener** (os::smart_ptr< **errorListener** > listener)

  Register listener.
- void **removeErrrorListener** (os::smart_ptr< **errorListener** > listener)

  Un-register listener.
- **errorPointer popError** ()

  Removes error from log.
- void **setLogLength** (unsigned int **logLength**)

  Set length of log.
- unsigned int **logLength** () const

  Return length of log.
- unsigned int **numberErrors** () const

  Return number of errors in log.

Protected Member Functions

- virtual void **logError** (**errorPointer** elm)

  Logs an error Dispatches an event to all listeners and stores the error in the log.

Private Attributes

- os::spinLock **listenerLock**

  Set protection mutex.
- os::smartSet< **errorListener** > **errorListen**

  Set of listeners.
- os::unsortedList< **error** > **errorLog**

  List of current errors.
- unsigned int **_logLength**

  Number of errors kept.

Friends

- class **errorListener**

  *Friendship with **crypto::errorListener** (p. 85).*

6.11.1 Detailed Description

Sends **crypto::error** (p. 81).

Sends and logs crypto:error pointers. Does not catch the errors, simply logs ones which have already been created and caught.

6.11.2 Constructor & Destructor Documentation

crypto::errorSender::errorSender ( ) [inline]

Error sender constructor.

Sets the length of the log to 20. Initializes with no errors and no listeners

virtual crypto::errorSender::~errorSender ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.11.3 Member Function Documentation

virtual void crypto::errorSender::logError (**errorPointer** elm) [protected], [virtual]

Logs an error Dispatches an event to all listeners and stores the error in the log.

Parameters

in	*elm*	Error to be logged

Returns

void

Reimplemented in **crypto::gateway** (p. 100).

unsigned int crypto::errorSender::logLength ( ) const [inline]

Return length of log.

Returns

crypto::errorSender::_logLength (p. 90)

unsigned int crypto::errorSender::numberErrors ( ) const [inline]

Return number of errors in log.

Returns

crypto::errorSender::errorLog.size()

errorPointer crypto::errorSender::popError ( )

Removes error from log.

Returns

Oldest recorded error

void crypto::errorSender::pushErrorListener (os::smart_ptr< **errorListener** > listener)

Register listener.

Parameters

[in/out]	listener Listener to register

Returns

void

void crypto::errorSender::removeErrrorListener (os::smart_ptr< **errorListener** > listener)

Un-register listener.

Parameters

`in`	*listener*	Listener to un-register

Returns

void

void crypto::errorSender::setLogLength (unsigned int logLength)

Set length of log.

Parameters

`in`	*logLength*	Target length of log

Returns

void

6.11.4 Friends And Related Function Documentation

friend class **errorListener** [friend]

Friendship with **crypto::errorListener** (p. 85).
The error listener must be able to add and remove itself from the sender's set.

6.11.5 Member Data Documentation

unsigned int crypto::errorSender::_logLength [private]

Number of errors kept.
Allows for old errors to expire in the event a sender logs a lot of errors.

os::smartSet<**errorListener**> crypto::errorSender::errorListen [private]

Set of listeners.
All of the listeners registered to this sender.

os::unsortedList<**error**> crypto::errorSender::errorLog [private]

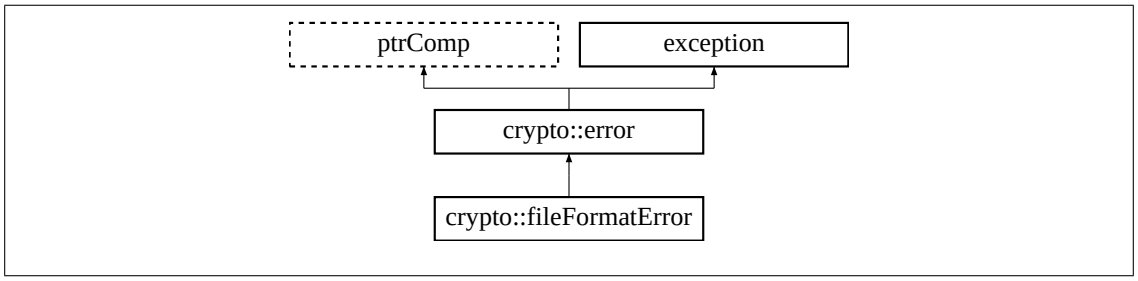
List of current errors.

os::spinLock crypto::errorSender::listenerLock [private]

Set protection mutex.
Protects access to the set of listeners, allows for multi-threading.

6.12 crypto::fileFormatError Class Reference

File format error.
Inheritance diagram for crypto::fileFormatError:



Public Member Functions

- virtual ~**fileFormatError** () throw ()
  Virtual destructor.
- std::string **errorTitle** () const
  Short error descriptor Returns "File Format Error".
- std::string **errorDescription** () const
  Long error descriptor Returns "The file is not of the specified format, and an error resulted".

6.12.1 Detailed Description

File format error.

Thrown when a file is parsed but an error occurs while parsing.

6.12.2 Constructor & Destructor Documentation

virtual crypto::fileFormatError::~fileFormatError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.12.3 Member Function Documentation

std::string crypto::fileFormatError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "The file is not of the specified format, and an error resulted".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::fileFormatError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "File Format Error".

Returns

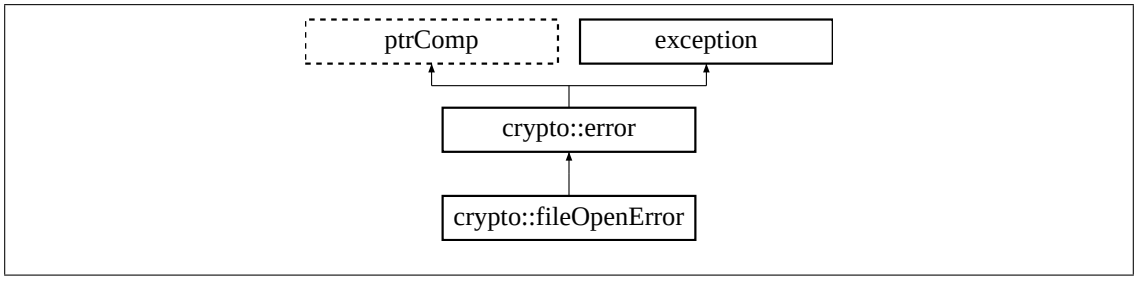
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.13 crypto::fileOpenError Class Reference

File open error.

Inheritance diagram for crypto::fileOpenError:



Public Member Functions

- virtual ~**fileOpenError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "File Open Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot open the specified file".

6.13.1 Detailed Description

File open error.

Thrown when a file cannot be found in the specified location.

6.13.2 Constructor & Destructor Documentation

virtual crypto::fileOpenError::~fileOpenError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.13.3 Member Function Documentation

std::string crypto::fileOpenError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot open the specified file".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::fileOpenError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "File Open Error".

Returns

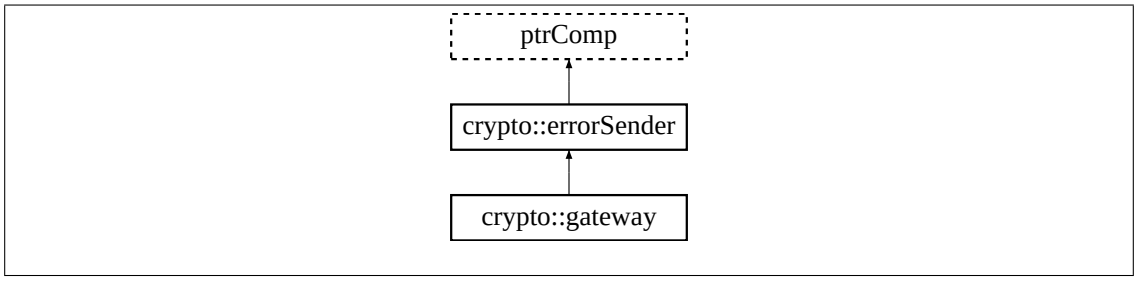
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.14 crypto::gateway Class Reference

Security gateway.

Inheritance diagram for crypto::gateway:



Public Member Functions

- **gateway** (os::smart_ptr< **user** > usr, std::string groupID="default")
  Gateway constructor.
- virtual ~**gateway** ()
  Virtual destructor.
- os::smart_ptr< **nodeGroup** > **brotherNode** ()
  Return the node group of the brother.
- os::smart_ptr< **message** > **getMessage** ()
  Returns next message from the gateway.
- os::smart_ptr< **message** > **send** (os::smart_ptr< **message** > msg)
  Send message through the gateway.
- os::smart_ptr< **message** > **ping** ()
  Ping message.
- os::smart_ptr< **message** > **processMessage** (os::smart_ptr< **message** > msg)
  Process incoming message.
- void **processTimestamps** ()
  Cycle time-stamp data.
- os::smart_ptr< **gatewaySettings** > **getBrotherSettings** ()
  Access brother settings.
- os::smart_ptr< **gatewaySettings** > **getSelfSettings** ()
  Access self settings.
- uint8_t **currentState** () const
  This gateway's status.
- uint8_t **brotherState** () const
  Brother gateway status.
- bool **secure** () const
  Gateway security established.
- uint64_t **timeout** () const
  Current receiver-side timeout value.
- uint64_t **safeTimeout** () const
  Current sender-side timeout value.
- uint64_t **errorTimeout** () const
  Current error timeout value.

- uint64_t **timeMessageReceived** () const

  Time-stamp of the last received message.
- uint64_t **timeMessageSent** () const

  Time-stamp of the last sent message.
- uint64_t **timeLastError** () const

  Time-stamp of the last error.

Static Public Attributes

- static const uint64_t **DEFAULT_TIMEOUT** =60

  Default timeout in seconds.
- static const uint64_t **DEFAULT_ERROR_TIMEOUT** =10

  Default error timeout in seconds.
- static const uint8_t **UNKNOWN_STATE** =0

  Unknown state value.
- static const uint8_t **UNKNOWN_BROTHER** =1

  Unknown brother state.
- static const uint8_t **SETTINGS_EXCHANGED** =2

  Settings exchanged state.
- static const uint8_t **ESTABLISHING_STREAM** =3

  Establishing stream state.
- static const uint8_t **STREAM_ESTABLISHED** =4

  Stream established state.
- static const uint8_t **SIGNING_STATE** =5

  Signing state.
- static const uint8_t **CONFIRM_OLD** =6

  Confirm old key state.
- static const uint8_t **ESTABLISHED** =7

  Stream established state.
- static const uint8_t **CONFIRM_ERROR_STATE** =252

  Confirm brother error state.
- static const uint8_t **BASIC_ERROR_STATE** =253

  Basic error state.
- static const uint8_t **TIMEOUT_ERROR_STATE** =254

  Timeout error state.
- static const uint8_t **PERMENANT_ERROR_STATE** =255

  Permanent error state.

Protected Member Functions

- void **logError** (**errorPointer** elm, uint8_t errType)

  Logs an error, with an error type.
- void **logError** (**errorPointer** elm)

  Logs an error, with type basic.

Private Member Functions

- void **clearStream** ()

  Resets stream tracking.
- void **buildStream** ()

  Builds the output stream.
- os::smart_ptr< **message** > **encrypt** (os::smart_ptr< **message** > msg)

  Encrypt a message.
- os::smart_ptr< **message** > **decrypt** (os::smart_ptr< **message** > msg)

  Decrypt a message.
- os::smart_ptr< **message** > **currentError** ()

  Build current error message.
- void **purgeLastError** ()

  Reset error.

Private Attributes

- os::smart_ptr< **gatewaySettings** > **selfSettings**

  Settings of this gateway.
- os::smart_ptr< **gatewaySettings** > **brotherSettings**

  Settings of the reciprocal gateway.
- os::spinLock **lock**

  Mutex protected gateway states.
- os::spinLock **stampLock**

  Mutex protecting timestamps.
- uint8_t **_currentState**

  Current state of this gateway.
- uint8_t **_brotherState**

  State of the reciprocal gateway.
- **errorPointer _lastError**

  Hold the most recent error.
- uint8_t **_lastErrorLevel**

  Holds the level of the last error.
- uint64_t **_errorTimestamp**

  Time-stamp of the last error.
- uint64_t **_timeout**

  Number of seconds till timeout.
- uint64_t **_safeTimeout**

  Number of seconds till partial timeout.
- uint64_t **_errorTimeout**

  Number of seconds for error timeout.
- uint64_t **_messageReceived**

  Time-stamp of last message received.
- uint64_t **_messageSent**

Time-stamp of last message sent.

- os::smart_ptr< **streamPackageFrame** > **selfStream**

  Stream algorithm for this gateway.
- os::smart_ptr< **publicKeyPackageFrame** > **selfPKFrame**

  Public key algorithm for this gateway.
- os::smart_ptr< **publicKey** > **selfPublicKey**

  Public/private key pair.
- os::smart_ptr< **number** > **selfPreciseKey**

  Public key for this gateway.
- os::smart_ptr< **streamPackageFrame** > **brotherStream**

  Stream algorithm for brother gateway.
- os::smart_ptr< **publicKeyPackageFrame** > **brotherPKFrame**

  Public key algorithm for bro.
- os::smart_ptr< **number** > **brotherPublicKey**

  Public key for brother gateway.
- os::smart_ptr< **message** > **streamMessageIn**

  Stream defining message: in.
- os::smart_ptr< **streamDecrypter** > **inputStream**

  Stream for incoming messages.
- uint64_t **streamEstTimestamp**

  Time the output stream was defined.
- os::smart_ptr< **message** > **streamMessageOut**

  Stream defining message: out.
- os::smart_ptr< **streamEncrypter** > **outputStream**

  Stream for outgoing messages.
- os::smart_ptr< uint8_t > **outputHashArray**

  Data for outgoing hashes.
- uint16_t **outputHashLength**

  Length of outgoing hash array.
- os::smart_ptr< **hash** > **selfPrimarySignatureHash**

  Hash for primary signature.
- os::smart_ptr< **hash** > **selfSecondarySignatureHash**

  Hash for historical signature.
- os::smart_ptr< **message** > **selfSigningMessage**

  Signing message: out.
- os::unsortedList< **hash** > **eligibleKeys**

  List of eligible public keys.
- os::smart_ptr< uint8_t > **inputHashArray**

  Data for incoming hashes.
- uint16_t **inputHashLength**

  Length of incoming hash array.
- os::smart_ptr< **hash** > **brotherPrimarySignatureHash**

  Hash of brother's primary signature.
- os::smart_ptr< **hash** > **brotherSecondarySignatureHash**

  Hash of brother's historical signature.

6.14.1 Detailed Description

Security gateway.

This gateway establishes a secured connection between two users. The connection uses the preferred algorithms as defined by the user.

6.14.2 Constructor & Destructor Documentation

crypto::gateway::gateway (os::smart_ptr< **user** > usr, std::string groupID = "default")

Gateway constructor.

Constructs a gateway from a user and a group ID. This initializes all gateway variables and binds the user settings to this gateway.

Parameters

in	*usr*	User sending information through this gateway
in	*groupID*	Defines group ID, "default" by default

virtual crypto::gateway::~gateway ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.14.3 Member Function Documentation

os::smart_ptr<**nodeGroup**> crypto::gateway::brotherNode ( )

Return the node group of the brother.

Uses the current key bank to find the node associated with this brother.

Returns

brother node

uint8_t crypto::gateway::brotherState ( ) const [inline]

Brother gateway status.

Returns

gateway::_brotherState (p. 102)

void crypto::gateway::buildStream ( ) [private]

Builds the output stream.

Returns

void

void crypto::gateway::clearStream ( ) [private]

Resets stream tracking.

Resets all pointers defined while establishing a secure stream.

Returns

void

os::smart_ptr<**message**> crypto::gateway::currentError ( ) [private]

Build current error message.

Returns

Message

uint8_t crypto::gateway::currentState ( ) const [inline]

This gateway's status.

Returns

gateway::_currentState (p. 102)

os::smart_ptr<**message**> crypto::gateway::decrypt (os::smart_ptr< **message** > msg) [private]

Decrypt a message.

Uses the established input stream to decrypt the provided message and return it as a new message.

Parameters

in	*msg*	Message to be decrypted

Returns

Decrypted message

os::smart_ptr<**message**> crypto::gateway::encrypt (os::smart_ptr< **message** > msg) [private]

Encrypt a message.

Uses the established output stream to encrypt the provided message and return it as a new message.

Parameters

in	*msg*	Message to be encrypted

Returns

Encrypted message

uint64_t crypto::gateway::errorTimeout ( ) const [inline]

Current error timeout value.

Returns

gateway::_errorTimeout (p. 102)

os::smart_ptr<**gatewaySettings**> crypto::gateway::getBrotherSettings ( ) [inline]

Access brother settings.

Returns

Pointer to brother settings

os::smart_ptr<**message**> crypto::gateway::getMessage ( )

Returns next message from the gateway.

The function only returns the next message from the gateway's perspective. Gateway management messages are returned by this function.

Returns

Next management message

os::smart_ptr<**gatewaySettings**> crypto::gateway::getSelfSettings ( ) [inline]

Access self settings.

Returns

Pointer to self settings

void crypto::gateway::logError (**errorPointer** elm, uint8_t errType) [protected]

Logs an error, with an error type.

Wraps the "logError" funciton as defined by the **crypto::errorSender** (p. 86) class, also sets this particular gateway into some error state.

Parameters

in	*elm*	Error description
in	*errType*	Error level to determine timeout

Returns

void

void crypto::gateway::logError (**errorPointer** elm) [inline], [protected], [virtual]

Logs an error, with type basic.
Sets this particular gateway into a default error state by calling "logError" with a type.

Parameters

in	*elm*	Eror description

Returns

void

Reimplemented from **crypto::errorSender** (p. 88).

os::smart_ptr<**message**> crypto::gateway::ping ( )

Ping message.
Returns the ping message as defined by the **gatewaySettings** (p. 107) in this gateway.

Returns

Ping message for this user

os::smart_ptr<**message**> crypto::gateway::processMessage (os::smart_ptr< **message** > msg)

Process incoming message.
Decrypts and processes an incoming message. Note that messages must be coming from the brother gateway of this gateway.

Parameters

in	*msg*	Message to be processed

Returns

Decrypted message

void crypto::gateway::processTimestamps ( )

Cycle time-stamp data.
Compares registered time-stamps with the current time to determine if any state changes need to be made.

Returns

void

void crypto::gateway::purgeLastError ( ) [private]

Reset error.

Resets all error variables and returns the gateway to its unconnected state.

Returns

void

uint64_t crypto::gateway::safeTimeout ( ) const [inline]

Current sender-side timeout value.

Returns

gateway::_safeTimeout (p. 103)

bool crypto::gateway::secure ( ) const [inline]

Gateway security established.

Returns

true if established, else, false

os::smart_ptr<**message**> crypto::gateway::send (os::smart_ptr< **message** > msg)

Send message through the gateway.

Takes a message and encrypts it with the gateway, assuming the secure stream has been established. Returns an encrypted version of the message sent through the gateway.

Parameters

in	*msg*	Message to be encrypted

Returns

Encrypted message

uint64_t crypto::gateway::timeLastError ( ) const [inline]

Time-stamp of the last error.

Returns

gateway::_errorTimestamp (p. 102)

uint64_t crypto::gateway::timeMessageReceived ( ) const [inline]

Time-stamp of the last received message.

Returns

gateway::_messageReceived (p. 102)

uint64_t crypto::gateway::timeMessageSent ( ) const [inline]

Time-stamp of the last sent message.

Returns

gateway::_messageSent (p. 103)

uint64_t crypto::gateway::timeout ( ) const [inline]

Current receiver-side timeout value.

Returns

gateway::_timeout (p. 103)

6.14.4 Member Data Documentation

uint8_t crypto::gateway::_brotherState [private]

State of the reciprocal gateway.

uint8_t crypto::gateway::_currentState [private]

Current state of this gateway.

uint64_t crypto::gateway::_errorTimeout [private]

Number of seconds for error timeout.

When dealing with a timeout error, this defines how many seconds to wait before allowing a connection again.

uint64_t crypto::gateway::_errorTimestamp [private]

Time-stamp of the last error.

errorPointer crypto::gateway::_lastError [private]

Hold the most recent error.

This holds logging information for the most recent serious error. If an error is thrown while in an error state, the more serious error is kept in this variable.

uint8_t crypto::gateway::_lastErrorLevel [private]

Holds the level of the last error.

Either Basic, timeout or permanent. These are 253, 254 and 255 respectively.

uint64_t crypto::gateway::_messageReceived [private]

Time-stamp of last message received.

uint64_t crypto::gateway::_messageSent [private]

Time-stamp of last message sent.

uint64_t crypto::gateway::_safeTimeout [private]

Number of seconds till partial timeout.
This value is used as the timeout value when sending messages and is less than the timeout value so that receiving is more permissive than sending.

uint64_t crypto::gateway::_timeout [private]

Number of seconds till timeout.
This value is used when calculating timeout for receiving messages.

const uint8_t crypto::gateway::BASIC_ERROR_STATE =253 [static]

Basic error state.
A gateway has logged a low-level error. The connection must be re-set and re-established.

os::smart_ptr<**publicKeyPackageFrame**> crypto::gateway::brotherPKFrame [private]

Public key algorithm for bro.

os::smart_ptr<**hash**> crypto::gateway::brotherPrimarySignatureHash [private]

Hash of brother's primary signature.
If this hash is defined, then this gateway's brother has properly signed with the public key it declared.

os::smart_ptr<**number**> crypto::gateway::brotherPublicKey [private]

Public key for brother gateway.

os::smart_ptr<**hash**> crypto::gateway::brotherSecondarySignatureHash [private]

Hash of brother's historical signature.
When this hash is defined, this gateway's brother has properly signed with a historical public key.

os::smart_ptr<**gatewaySettings**> crypto::gateway::brotherSettings [private]

Settings of the reciprocal gateway.
Defined by the ping message which is received by this gateway's brother gateway.

os::smart_ptr<**streamPackageFrame**> crypto::gateway::brotherStream [private]

Stream algorithm for brother gateway.

const uint8_t crypto::gateway::CONFIRM_ERROR_STATE =252 [static]

Confirm brother error state.

In this state, a gateway is acknowledging to it's brother that the error notification sent by the brother was received and logged.

const uint8_t crypto::gateway::CONFIRM_OLD =6 [static]

Confirm old key state.

This indicates that a gateway has authenticated the identity of it's brother but has not been notified that its identity has been authenticated.

const uint64_t crypto::gateway::DEFAULT_ERROR_TIMEOUT =10 [static]

Default error timeout in seconds.

const uint64_t crypto::gateway::DEFAULT_TIMEOUT =60 [static]

Default timeout in seconds.

os::unsortedList<**hash**> crypto::gateway::eligibleKeys [private]

List of eligible public keys.

This list of hashes comes from the brother of this gateway. It is a list of the hashes of public keys associated with this node.

const uint8_t crypto::gateway::ESTABLISHED =7 [static]

Stream established state.

A secure and authentic stream has been established. Messages can be passed securely through the gateway.

const uint8_t crypto::gateway::ESTABLISHING_STREAM =3 [static]

Establishing stream state.

In this state, a gateway sends a symmetric stream key encrypted with the public key of the brother gateway.

os::smart_ptr<uint8_t> crypto::gateway::inputHashArray [private]

Data for incoming hashes.

uint16_t crypto::gateway::inputHashLength [private]

Length of incoming hash array.

os::smart_ptr<**streamDecrypter**> crypto::gateway::inputStream [private]

Stream for incoming messages.

os::spinLock crypto::gateway::lock [private]

Mutex protected gateway states.

os::smart_ptr<uint8_t> crypto::gateway::outputHashArray [private]

Data for outgoing hashes.

uint16_t crypto::gateway::outputHashLength [private]

Length of outgoing hash array.

os::smart_ptr<**streamEncrypter**> crypto::gateway::outputStream [private]

Stream for outgoing messages.

const uint8_t crypto::gateway::PERMENANT_ERROR_STATE =255 [static]

Permanent error state.

When gateways are in this state, a catastrophic error has occurred and the gateway refuses to reconnect.

os::smart_ptr<**publicKeyPackageFrame**> crypto::gateway::selfPKFrame [private]

Public key algorithm for this gateway.

os::smart_ptr<**number**> crypto::gateway::selfPreciseKey [private]

Public key for this gateway.

os::smart_ptr<**hash**> crypto::gateway::selfPrimarySignatureHash [private]

Hash for primary signature.

os::smart_ptr<**publicKey**> crypto::gateway::selfPublicKey [private]

Public/private key pair.

os::smart_ptr<**hash**> crypto::gateway::selfSecondarySignatureHash [private]

Hash for historical signature.

os::smart_ptr<**gatewaySettings**> crypto::gateway::selfSettings [private]

Settings of this gateway.

Defined by the user which constructed this gateway.

os::smart_ptr<**message**> crypto::gateway::selfSigningMessage [private]

Signing message: out.

This is a record of the message which was used to sign the current and historical public keys by this gateway in order to minimize the number of public key operations preformed.

os::smart_ptr<**streamPackageFrame**> crypto::gateway::selfStream [private]

Stream algorithm for this gateway.

const uint8_t crypto::gateway::SETTINGS_EXCHANGED =2 [static]

Settings exchanged state.

Indicates that a gateway has received a ping message from its reciprocal gateway, but has not received notification that the reciprocal gateway has received the ping message from this gateway.

const uint8_t crypto::gateway::SIGNING_STATE =5 [static]

Signing state.

Gateways in this state have established a secure stream with their brother node and now need to prove they have access to their declared public key. The signing message also contains hashes of keys associated with the particular node.

os::spinLock crypto::gateway::stampLock [private]

Mutex protecting timestamps.

const uint8_t crypto::gateway::STREAM_ESTABLISHED =4 [static]

Stream established state.

Gateways in this state continue to send the symmetric stream key, but also indicates to the brother gateway that the stream key sent by it has been received.

uint64_t crypto::gateway::streamEstTimestamp [private]

Time the output stream was defined.

Allows for redefinition of the output stream if the definition becomes stale.

os::smart_ptr<**message**> crypto::gateway::streamMessageIn [private]

Stream defining message: in.

This is a record of the message which defined the incoming stream in-order to minimize public key cryptography performed.

os::smart_ptr<**message**> crypto::gateway::streamMessageOut [private]

Stream defining message: out.

This is a record of the message which defined the outgoing stream in-order to minimize public key cryptography performed.

const uint8_t crypto::gateway::TIMEOUT_ERROR_STATE =254 [static]

Timeout error state.

Gateways are placed in this state when an error occurs while authenticating the connection. Because an error in this state is usually both expensive and indicative of unauthorized access, when errors occur, this state forces a certain amount of time in the error state before allowing reconnection.

const uint8_t crypto::gateway::UNKNOWN_BROTHER =1 [static]

Unknown brother state.

A gateway is in this state when it is unaware of the gateway settings of its reciprocal, or brother, gateway. In short, a gateway which does not know its brother has not received a ping.

const uint8_t crypto::gateway::UNKNOWN_STATE =0 [static]

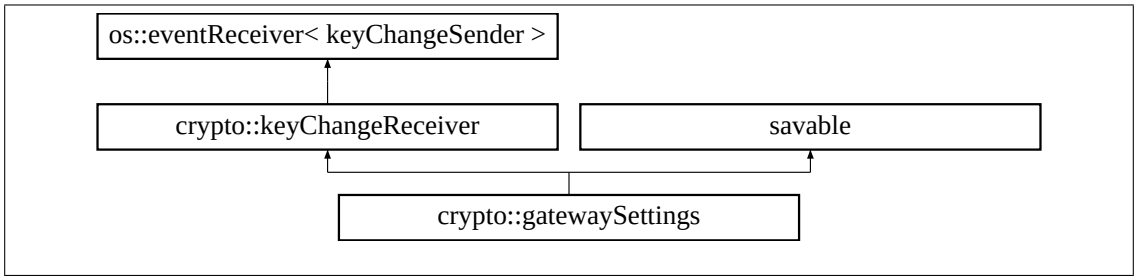
Unknown state value.

This state is used by a gateway when the it is not aware of the current state of its reciprocal gateway. A gateway should never be in this state itself.

6.15 crypto::gatewaySettings Class Reference

Holds settings for gateway encryption.

Inheritance diagram for crypto::gatewaySettings:



Public Member Functions

- **gatewaySettings** (os::smart_ptr< **user** > usr, std::string **groupID**, std::string **filePath**="")

  User constructor.
- **gatewaySettings** (const **message** &msg)

  Ping message constructor.
- virtual ~**gatewaySettings** ()

  Virtual destructor.
- os::smartXMLNode **generateSaveTree** ()

  Generate XML save stream.
- void **update** ()

  Ensure preferred algorithms are defined.
- void **save** ()

Saves the class to a file Saves the settings to an XML file, if the file path is defined.

- void **load** ()

  Loads the class from a file Loads the settings from an XML file, if the file path is defined.
- const std::string & **filePath** () const

  Return reference to the file path.
- const std::string & **groupID** () const

  Return reference to the group ID.
- const std::string & **nodeName** () const

  Return reference to the node name.
- os::smart_ptr< **user** > **getUser** ()

  Return user, if it is defined.
- os::smart_ptr< **publicKey** > **getPrivateKey** ()

  Return public/private key pair, if it is defined.
- os::smart_ptr< **number** > **getPublicKey** ()

  Return public key.
- uint16_t **prefferedPublicKeyAlgo** () const

  Return public key algorithm ID.
- uint16_t **prefferedPublicKeySize** () const

  Return public key algorithm size.
- uint16_t **prefferedHashAlgo** () const

  Return hash algorithm ID.
- uint16_t **prefferedHashSize** () const

  Return hash size.
- uint16_t **prefferedStreamAlgo** () const

  Return stream algorithm ID.
- os::smart_ptr< **message** > **ping** ()

  Construct a ping message.
- bool **operator==** (const **gatewaySettings** &cmp) const

  Equality comparison operator.
- bool **operator!=** (const **gatewaySettings** &cmp) const

  Not-equals comparison operator.
- bool **operator**< (const **gatewaySettings** &cmp) const

  Less-than comparison operator.
- bool **operator**> (const **gatewaySettings** &cmp) const

  Greater-than comparison operator.
- bool **operator**<**=** (const **gatewaySettings** &cmp) const

  Less-than/Equals-to comparison operator.
- bool **operator**>**=** (const **gatewaySettings** &cmp) const

  Greater-than/Equals-to comparison operator.

Public Attributes

- os::multiLock **lock**

  Read/write mutex.

Protected Member Functions

- void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggered when the public key is changed.

Private Attributes

- std::string **_groupID**

  Group ID of the node, unique to this settings class.
- std::string **_nodeName**

  Name of the node, defined by the user.
- std::string **_filePath**

  Save file path.
- os::smart_ptr< **user** > **_user**

  Pointer to the user class.
- os::smart_ptr< **publicKey** > **_privateKey**

  Pointer to public/private key pair.
- os::smart_ptr< **number** > **_publicKey**

  Pointer to the public key.
- uint16_t **_prefferedPublicKeyAlgo**

  Public key algorithm ID.
- uint16_t **_prefferedPublicKeySize**

  Public key size (uint32_t size)
- uint16_t **_prefferedHashAlgo**

  Hash algorithm ID.
- uint16_t **_prefferedHashSize**

  Hash size (in bytes)
- uint16_t **_prefferedStreamAlgo**

  Stream algorithm ID.

6.15.1 Detailed Description

Holds settings for gateway encryption.

Contains all of the information needed to define how the gateway functions. This includes which algorithms are white-listed, which are black- listed and which are preferred. Note that this settings class can define the settings for a node whose private key is known or for a node whose private key is unknown.

6.15.2 Constructor & Destructor Documentation

crypto::gatewaySettings::gatewaySettings (os::smart_ptr< **user** > usr, std::string groupID, std::string filePath = "")

User constructor.

Constructs the class from a user. While this constructor can be called outside the user class, it is suggested to use the interface provided in **crypto::user** (p. 255) to create new gateway settings.

Parameters

in	*usr*	User defining the settings
in	*groupID*	Group ID of the settings
in	*filePath*	Save file location (optional)

crypto::gatewaySettings::gatewaySettings (const **message** & msg)

Ping message constructor.

Constructs the gateway settings from a ping message. This is usually used by the gateway to parse ping messages it receives.

Parameters

in	*msg*	Ping message

virtual crypto::gatewaySettings::~gatewaySettings ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.15.3 Member Function Documentation

const std::string& crypto::gatewaySettings::filePath ( ) const [inline]

Return reference to the file path.

Returns

gatewaySettings::_filePath (p. 114)

os::smartXMLNode crypto::gatewaySettings::generateSaveTree ( )

Generate XML save stream.

Returns

XML save tree

os::smart_ptr<**publicKey**> crypto::gatewaySettings::getPrivateKey ( ) [inline]

Return public/private key pair, if it is defined.

Returns

gatewaySettings::_privateKey (p. 115)

os::smart_ptr<**number**> crypto::gatewaySettings::getPublicKey ( ) `[inline]`

Return public key.

Returns

gatewaySettings::_publicKey (p. 115)

os::smart_ptr<**user**> crypto::gatewaySettings::getUser ( ) `[inline]`

Return user, if it is defined.

Returns

gatewaySettings::_user (p. 115)

const std::string& crypto::gatewaySettings::groupID ( ) const `[inline]`

Return reference to the group ID.

Returns

gatewaySettings::_groupID (p. 115)

void crypto::gatewaySettings::load ( )

Loads the class from a file Loads the settings from an XML file, if the file path is defined.

Returns

void

const std::string& crypto::gatewaySettings::nodeName ( ) const `[inline]`

Return reference to the node name.

Returns

gatewaySettings::_nodeName (p. 115)

bool crypto::gatewaySettings::operator!= (const **gatewaySettings** & cmp) const `[inline]`

Not-equals comparison operator.
Uses the group ID to gateway settings.

Parameters

`in`	*cmp*	Object to compare against

Returns

this->_groupID != cmp._groupID

bool crypto::gatewaySettings::operator< (const **gatewaySettings** & cmp) const [inline]

Less-than comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID < cmp._groupID

bool crypto::gatewaySettings::operator<= (const **gatewaySettings** & cmp) const [inline]

Less-than/Equals-to comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID <= cmp._groupID

bool crypto::gatewaySettings::operator== (const **gatewaySettings** & cmp) const [inline]

Equality comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID == cmp._groupID

bool crypto::gatewaySettings::operator> (const **gatewaySettings** & cmp) const [inline]

Greater-than comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID > cmp._groupID

bool crypto::gatewaySettings::operator>= (const **gatewaySettings** & cmp) const [inline]

Greater-than/Equals-to comparison operator.
Uses the group ID to gateway settings.

Parameters

in	*cmp*	Object to compare against

Returns

this->_groupID >= cmp._groupID

os::smart_ptr<**message**> crypto::gatewaySettings::ping ( )

Construct a ping message.

Returns

New ping message

uint16_t crypto::gatewaySettings::prefferedHashAlgo ( ) const [inline]

Return hash algorithm ID.

Returns

gatewaySettings::_prefferedHashAlgo (p. 115)

uint16_t crypto::gatewaySettings::prefferedHashSize ( ) const [inline]

Return hash size.

Returns

gatewaySettings::_prefferedHashSize (p. 115)

uint16_t crypto::gatewaySettings::prefferedPublicKeyAlgo ( ) const [inline]

Return public key algorithm ID.

Returns

gatewaySettings::_prefferedPublicKeyAlgo (p. 115)

uint16_t crypto::gatewaySettings::prefferedPublicKeySize ( ) const [inline]

Return public key algorithm size.

Returns

gatewaySettings::_prefferedPublicKeySize (p. 115)

uint16_t crypto::gatewaySettings::prefferedStreamAlgo ( ) const [inline]

Return stream algorithm ID.

Returns

gatewaySettings::_prefferedStreamAlgo (p. 115)

void crypto::gatewaySettings::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) [protected], [virtual]

Triggered when the public key is changed.

Updates the gateway settings when the user indicates a public key has been updated.

Parameters

in	*pbk*	Updated public/private key pair

Returns

void

Reimplemented from **crypto::keyChangeReceiver** (p. 145).

void crypto::gatewaySettings::save ( )

Saves the class to a file Saves the settings to an XML file, if the file path is defined.

Returns

void

void crypto::gatewaySettings::update ( )

Ensure preferred algorithms are defined.

Uses current information in the class to determine if known algorithms define the preferred algorithms in this class. If the preferred algorithms are not defined, they are changed to defined algorithms.

Returns

void

6.15.4 Member Data Documentation

std::string crypto::gatewaySettings::_filePath [private]

Save file path.

If the setting was defined by the user and not a "ping" message, it will often have a save file location.

std::string crypto::gatewaySettings::_groupID [private]

Group ID of the node, unique to this settings class.

std::string crypto::gatewaySettings::_nodeName [private]

Name of the node, defined by the user.

uint16_t crypto::gatewaySettings::_prefferedHashAlgo [private]

Hash algorithm ID.

uint16_t crypto::gatewaySettings::_prefferedHashSize [private]

Hash size (in bytes)

uint16_t crypto::gatewaySettings::_prefferedPublicKeyAlgo [private]

Public key algorithm ID.

uint16_t crypto::gatewaySettings::_prefferedPublicKeySize [private]

Public key size (uint32_t size)

uint16_t crypto::gatewaySettings::_prefferedStreamAlgo [private]

Stream algorithm ID.

os::smart_ptr<**publicKey**> crypto::gatewaySettings::_privateKey [private]

Pointer to public/private key pair.

os::smart_ptr<**number**> crypto::gatewaySettings::_publicKey [private]

Pointer to the public key.

os::smart_ptr<**user**> crypto::gatewaySettings::_user [private]

Pointer to the user class.

os::multiLock crypto::gatewaySettings::lock

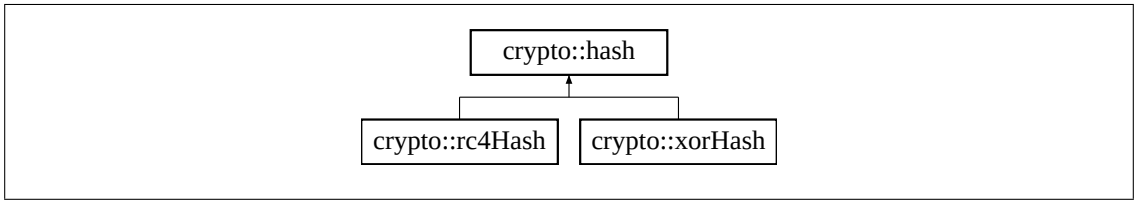
Read/write mutex.

When this class is defined by a user, it is possible for the user to change the gateway settings during runtime. Because of this, a read/write lock is required.

6.16 crypto::hash Class Reference

Base hash class.

Inheritance diagram for crypto::hash:



Public Member Functions

- **hash** (const **hash** &cpy)

  Hash copy constructor.
- **hash** & **operator=** (const **hash** &cpy)

  Equality constructor.
- virtual ~**hash** ()

  Virtual destructor.
- int **compare** (const **hash** *_comp) const

  Comparison function.
- virtual void **preformHash** (unsigned char ***data**, uint32_t dLen)

  Binds a data-set.
- virtual std::string **algorithmName** () const

  Algorithm name string access.
- uint16_t **algorithm** () const

  Current algorithm ID.
- uint16_t **size** () const

  Current hash size.
- uint32_t **numBits** () const

  Current hash size, bits.
- unsigned char * **data** ()

  Modifiable data access.
- const unsigned char * **data** () const

  Constant data access.
- unsigned char **operator[ ]** (uint16_t pos) const

  Modifiable data access.
- unsigned char & **operator[ ]** (uint16_t pos)

  Constant data access.
- std::string **toString** () const

  Converts hash to string.
- void **fromString** (const std::string &str)

  Converts from string.

- bool **operator==** (const **hash** &comp) const
- bool **operator!=** (const **hash** &comp) const
- bool **operator**> (const **hash** &comp) const
- bool **operator**>**=** (const **hash** &comp) const
- bool **operator**< (const **hash** &comp) const
- bool **operator**<**=** (const **hash** &comp) const

Static Public Member Functions

- static std::string **staticAlgorithmName** ()

  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()

  Algorithm ID number access.

Protected Member Functions

- **hash** (uint16_t **algorithm**=algo::hashNULL, uint16_t **size**=size::defaultHash)

  Default hash constructor.

Protected Attributes

- uint16_t **_size**

  Number of bytes in the hash.
- unsigned char * **_data**

  Raw hash data.

Private Attributes

- uint16_t **_algorithm**

  Hash algorithm ID.

6.16.1 Detailed Description

Base hash class.

This class manages the raw data of all hashes. Subsequent hashes define different algorithms to populate the hashes.

6.16.2 Constructor & Destructor Documentation

crypto::hash::hash (uint16_t algorithm = `algo::hashNULL`, uint16_t size = `size::defaultHash`) `[protected]`

Default hash constructor.

Constructs a hash with the given size and algorithm ID, initializing the entire hash itself to 0.

Parameters

in	*algorithm*	Algorithm ID, NULL by default
in	*size*	Size of hash, crypto::size::defaultHash by default

crypto::hash::hash (const **hash** & cpy)

Hash copy constructor.

Constructs a hash with a hash. This copy constructor re-initializes the data array for the new hash.

Parameters

in	*cpy*	Hash to copy

virtual crypto::hash::~hash ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.16.3 Member Function Documentation

uint16_t crypto::hash::algorithm ( ) const [inline]

Current algorithm ID.

Returns the algorithm ID bound to this hash.

Returns

crypto::hash::_algorithm (p. 121)

virtual std::string crypto::hash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.

Returns the name of the current algorithm string. This function is virtual, so changes for each hash algorithm

Returns

"NULL"

Reimplemented in **crypto::xorHash** (p. 270), and **crypto::rc4Hash** (p. 239).

int crypto::hash::compare (const **hash** * _comp) const

Comparison function.

Takes into consideration the algorithm, size of the data and content of the hash. Used for all of the equality operators.

Returns

0 if equal, 1 if greater than, -1 if less than

unsigned char* crypto::hash::data ( ) [inline]

Modifiable data access.
Provides mutable data-access to the raw hash data.

Returns

crypto::hash::_data (p. 121)

const unsigned char* crypto::hash::data ( ) const [inline]

Constant data access.
Provides immutable data-access to the raw hash data.

Returns

crypto::hash::_data (p. 121)

void crypto::hash::fromString (const std::string & str)

Converts from string.
Rebuilds the hash from a hex string.

Parameters

in	*str*	Hex string

Returns

String representation of the hash

uint32_t crypto::hash::numBits ( ) const [inline]

Current hash size, bits.
Return the hash size bound to this hash in bits.

Returns

crypto::hash::_size (p. 121)*8

bool crypto::hash::operator!= (const **hash** & comp) const [inline]

bool crypto::hash::operator< (const **hash** & comp) const [inline]

bool crypto::hash::operator<= (const **hash** & comp) const [inline]

hash& crypto::hash::operator= (const **hash** & cpy)

Equality constructor.
Rebuild this hash with the data from another hash.

Parameters

in	*cpy*	Hash to copy

Returns

Reference to this

bool crypto::hash::operator== (const **hash** & comp) const [inline]

bool crypto::hash::operator> (const **hash** & comp) const [inline]

bool crypto::hash::operator>= (const **hash** & comp) const [inline]

unsigned char crypto::hash::operator[ ] (uint16_t pos) const

Modifiable data access.

Provides mutable data-access to the raw hash data.

Parameters

in	*pos*	Data index

Returns

crypto::hash::_data (p. 121)[pos]

unsigned char& crypto::hash::operator[ ] (uint16_t pos)

Constant data access.

Provides immutable data-access to the raw hash data.

Parameters

in	*pos*	Data index

Returns

crypto::hash::_data (p. 121)[pos]

virtual void crypto::hash::preformHash (unsigned char * data, uint32_t dLen) [inline], [virtual]

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

uint16_t crypto::hash::size ( ) const [inline]

Current hash size.
Returns the hash size bound to this hash in bytes.

Returns

crypto::hash::_size (p. 121)

static uint16_t crypto::hash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.
Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashNULL

static std::string crypto::hash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.
Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"NULL"

std::string crypto::hash::toString ( ) const

Converts hash to string.
Converts the hash to a hex string.

Returns

String representation of the hash

6.16.4 Member Data Documentation

uint16_t crypto::hash::_algorithm [private]

Hash algorithm ID.

unsigned char* crypto::hash::_data [protected]

Raw hash data.

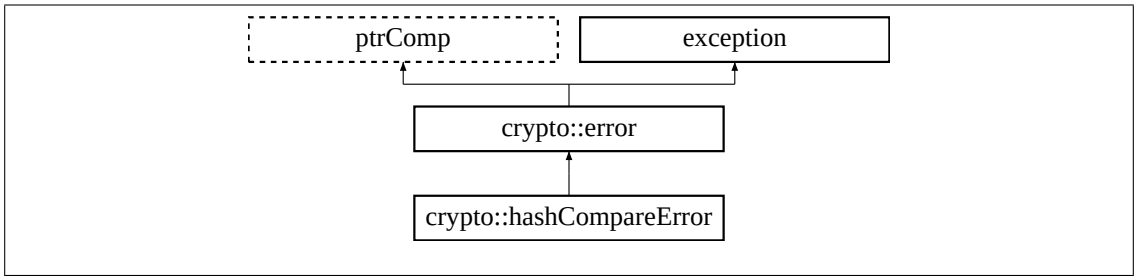
uint16_t crypto::hash::_size [protected]

Number of bytes in the hash.

6.17 crypto::hashCompareError Class Reference

Hash mis-match.

Inheritance diagram for crypto::hashCompareError:



Public Member Functions

- virtual ~**hashCompareError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Hash Compare".
- std::string **errorDescription** () const

  Long error descriptor Returns "Provided and calculated hashes do not match".

6.17.1 Detailed Description

Hash mis-match.

Thrown when two hashes do not match. This error can be indicative of larger security issues, as it most commonly occurs during a failed authentication.

6.17.2 Constructor & Destructor Documentation

virtual crypto::hashCompareError::~hashCompareError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.17.3 Member Function Documentation

std::string crypto::hashCompareError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Provided and calculated hashes do not match".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::hashCompareError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Hash Compare".

Returns

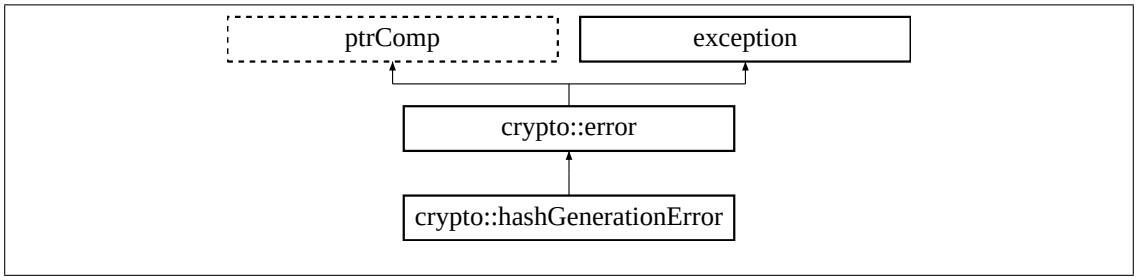
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.18 crypto::hashGenerationError Class Reference

Hash generation error.

Inheritance diagram for crypto::hashGenerationError:



Public Member Functions

- virtual ~**hashGenerationError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Hash Generation".
- std::string **errorDescription** () const

  Long error descriptor Returns "Could not generate a hash with the given arguments".

6.18.1 Detailed Description

Hash generation error.

Thrown when a hash encounters an error while being created.

6.18.2 Constructor & Destructor Documentation

virtual crypto::hashGenerationError::~hashGenerationError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.18.3 Member Function Documentation

std::string crypto::hashGenerationError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Could not generate a hash with the given arguments".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::hashGenerationError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Hash Generation".

Returns

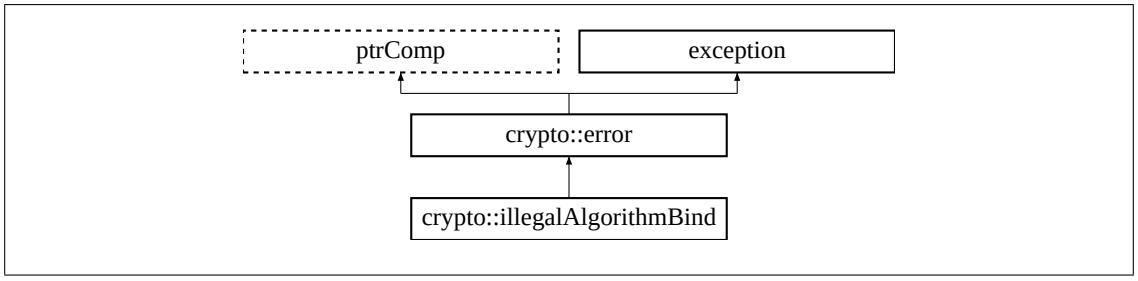
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.19 crypto::illegalAlgorithmBind Class Reference

Algorithm bound failure.

Inheritance diagram for crypto::illegalAlgorithmBind:



Public Member Functions

- **illegalAlgorithmBind** (std::string algoName)

  Illegal algorithm error.
- virtual ~**illegalAlgorithmBind** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Illegal Algorithm Bind".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot bind algorithm of type: <algorithmName>".

Private Attributes

- std::string **algorithmName**

  Name of algorithm.

6.19.1 Detailed Description

Algorithm bound failure.

Thrown when an algorithm cannot be found or used. Usually indicates the specified algorithm is not defined by the active version.

6.19.2 Constructor & Destructor Documentation

crypto::illegalAlgorithmBind::illegalAlgorithmBind (std::string algoName) `[inline]`

Illegal algorithm error.

Parameters

`in`	*algoName*	Name of illegal algorithm

virtual crypto::illegalAlgorithmBind::~illegalAlgorithmBind ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.19.3 Member Function Documentation

std::string crypto::illegalAlgorithmBind::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Cannot bind algorithm of type: <algorithmName>".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::illegalAlgorithmBind::errorTitle ( ) const `[inline]`, `[virtual]`

Short error descriptor Returns "Illegal Algorithm Bind".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.19.4 Member Data Documentation

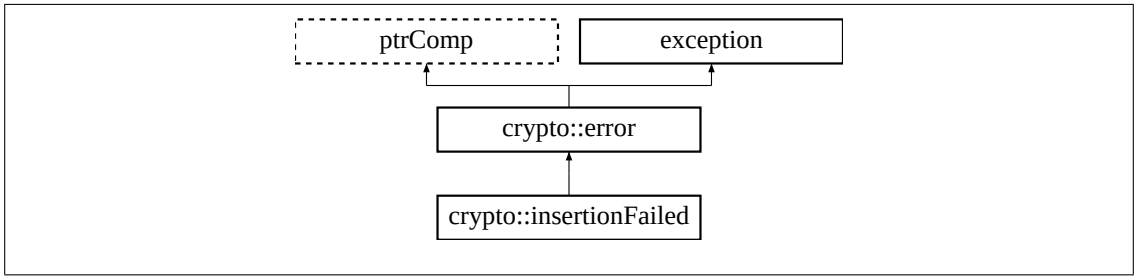
std::string crypto::illegalAlgorithmBind::algorithmName [private]

Name of algorithm.

6.20 crypto::insertionFailed Class Reference

ADS Insertion Failed.

Inheritance diagram for crypto::insertionFailed:



Public Member Functions

- virtual ~**insertionFailed** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Insertion Failed".
- std::string **errorDescription** () const

  Long error descriptor Returns "Insertion into an abstract data-structure unexpectedly failed".

6.20.1 Detailed Description

ADS Insertion Failed.

Thrown when insertion to an os::ads structure unexpectedly fails.

6.20.2 Constructor & Destructor Documentation

virtual crypto::insertionFailed::~insertionFailed ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.20.3 Member Function Documentation

std::string crypto::insertionFailed::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Insertion into an abstract data-structure unexpectedly failed".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::insertionFailed::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Insertion Failed".

Returns

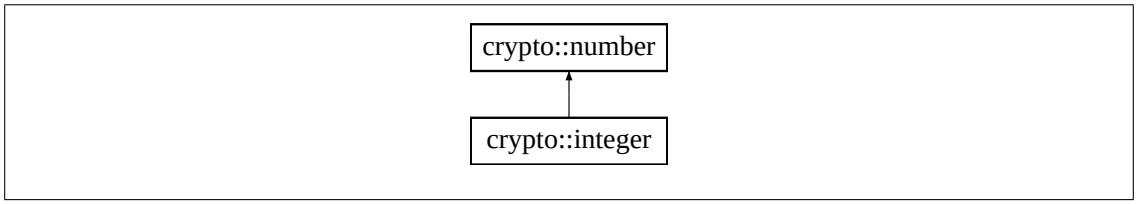
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.21 crypto::integer Class Reference

Integer number definition.

Inheritance diagram for crypto::integer:



Public Member Functions

- **integer** ()

  Default integer constructor.
- **integer** (uint16_t **size**)

  Construct integer with size.
- **integer** (const uint32_t *d, uint16_t **size**)

  Construct integer with data array.
- **integer** (const **integer** &num)

  Copy constructor.
- virtual ~**integer** ()

  Virtual destructor.
- bool **checkType** () const

  Check if the number is valid.
- **operator number** () const

  Allows integer to be cast as a number.
- **integer operator+** (const **integer** &n) const

  Integer addition operator.
- **integer** & **operator+=** (const **integer** &n)

Integer addition equals operator.

- **integer** & **operator++** ()

  Increment operator.
- **integer operator++** (int dummy)

  Increment operator.
- **integer operator-** (const **integer** &n) const

  Integer subtraction operator.
- **integer** & **operator-=** (const **integer** &n)

  Integer subtraction equals operator.
- **integer** & **operator--** ()

  Decrement operator.
- **integer operator--** (int dummy)

  Decrement operator.
- **integer operator**>> (uint16_t n) const

  Right shift operator.
- **integer operator**<< (uint16_t n) const

  Left shift operator.
- **integer operator**∗ (const **integer** &n) const

  Integer multiplication operator.
- **integer** & **operator**∗**=** (const **integer** &n)

  Integer multiplication equals operator.
- **integer operator**/ (const **integer** &n) const

  Integer division operator.
- **integer** & **operator/=** (const **integer** &n)

  Integer division equals operator.
- **integer operator%** (const **integer** &n) const

  Integer modulo operator.
- **integer** & **operator%=** (const **integer** &n)

  Integer modulo equals operator.
- **integer exponentiation** (const **integer** &n) const

  Integer exponentiation function.
- **integer** & **exponentiationEquals** (const **integer** &n)

  Integer exponentiation equals function.
- **integer moduloExponentiation** (const **integer** &n, const **integer** &mod) const

  Integer modulo-exponentiation function.
- **integer** & **moduloExponentiationEquals** (const **integer** &n, const **integer** &mod)

  Integer modulo-exponentiation equals function.
- **integer gcd** (const **integer** &n) const

  Integer GCD function.
- **integer** & **gcdEquals** (const **integer** &n)

  Integer GCD equals function.
- **integer modInverse** (const **integer** &m) const

  Integer modular inverse function.

- **integer** & **modInverseEquals** (const **integer** &n)

  Integer modular inverse equals function.
- bool **prime** (uint16_t testVal=algo::primeTestCycle) const

  Test if this integer is prime.

Static Public Member Functions

- static **integer zero** ()

  Constructs a '0' integer.
- static **integer one** ()

  Constructs a '1' integer.
- static **integer two** ()

  Constructs a '2' integer.

Additional Inherited Members

6.21.1 Detailed Description

Integer number definition.
A traditional numerical definition which can be of arbitrary size.

6.21.2 Constructor & Destructor Documentation

crypto::integer::integer ( )

Default integer constructor.

crypto::integer::integer (uint16_t size)

Construct integer with size.

Parameters

in	*size*	Size integer is initialized with

crypto::integer::integer (const uint32_t * d, uint16_t size)

Construct integer with data array.

Parameters

in	*d*	Data array to be bound
in	*size*	Size of array

crypto::integer::integer (const **integer** & num)

Copy constructor.

Parameters

in	*num*	Integer used to construct this

virtual crypto::integer::~integer ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.21.3 Member Function Documentation

bool crypto::integer::checkType ( ) const [virtual]

Check if the number is valid.

Checks to ensure that the number definition for this object is the Base-10 type. Ensure that all basic mathematical operators are defined.

Returns

true if valid type, else, false

Reimplemented from **crypto::number** (p. 182).

integer crypto::integer::exponentiation (const **integer** & n) const

Integer exponentiation function.

Parameters

in	*n*	Integer to be raised to

Returns

this^n

integer& crypto::integer::exponentiationEquals (const **integer** & n)

Integer exponentiation equals function.

Parameters

in	*n*	Integer to be raised to

Returns

this = this^n

integer crypto::integer::gcd (const **integer** & n) const

Integer GCD function.

Parameters

in	*n*	Integer to be compared against

Returns

GCD of this and n

integer& crypto::integer::gcdEquals (const **integer** & n)

Integer GCD equals function.

Parameters

in	*n*	Integer to be compared against

Returns

this = GCD of this and n

integer crypto::integer::modInverse (const **integer** & m) const

Integer modular inverse function.

Parameters

in	*n*	Integer representing modulo space

Returns

(this^-1) % n

integer& crypto::integer::modInverseEquals (const **integer** & n)

Integer modular inverse equals function.

Parameters

in	*n*	Integer representing modulo space

Returns

this = (this^-1) % n

integer crypto::integer::moduloExponentiation (const **integer** & n, const **integer** & mod) const

Integer modulo-exponentiation function.

Parameters

in	*n*	Integer to be raised to
in	*mod*	Integer representing modulo space

Returns

this^n % mod

integer& crypto::integer::moduloExponentiationEquals (const **integer** & n, const **integer** & mod)

Integer modulo-exponentiation equals function.

Parameters

in	*n*	Integer to be raised to
in	*mod*	Integer representing modulo space

Returns

this = this^n % mod

static **integer** crypto::integer::one ( ) [static]

Constructs a '1' integer.

Returns

1

crypto::integer::operator **number** ( ) const [inline]

Allows integer to be cast as a number.

Returns

number(∗this)

integer crypto::integer::operator% (const **integer** & n) const

Integer modulo operator.

Parameters

in	*n*	Integer defining modulo space this % n

integer& crypto::integer::operator%= (const **integer** & n)

Integer modulo equals operator.

Parameters

in	*n*	Integer defining modulo space this = this % n

integer crypto::integer::operator∗ (const **integer** & n) const

Integer multiplication operator.

Parameters

in	*n*	Integer to be multiplied this ∗ n

integer& crypto::integer::operator∗= (const **integer** & n)

Integer multiplication equals operator.

Parameters

in	*n*	Integer to be multiplied this = this ∗ n

integer crypto::integer::operator+ (const **integer** & n) const

Integer addition operator.

Parameters

in	*n*	Integer to be added this + n

integer& crypto::integer::operator++ ( )

Increment operator.

Returns

this++

integer crypto::integer::operator++ (int dummy)

Increment operator.

Returns

++this

integer& crypto::integer::operator+= (const **integer** & n)

Integer addition equals operator.

Parameters

in	*n*	Integer to be added this = this + n

integer crypto::integer::operator- (const **integer** & n) const

Integer subtraction operator.

Parameters

in	*n*	Integer to be subtracted this - n

integer& crypto::integer::operator-- ( )

Decrement operator.

Returns

this–

integer crypto::integer::operator-- (int dummy)

Decrement operator.

Returns

–this

integer& crypto::integer::operator-= (const **integer** & n)

Integer subtraction equals operator.

Parameters

in	*n*	Integer to be subtracted this = this - n

integer crypto::integer::operator/ (const **integer** & n) const

Integer division operator.

Parameters

in	*n*	Integer to be divided by this / n

integer& crypto::integer::operator/= (const **integer** & n)

Integer division equals operator.

Parameters

in	*n*	Integer to be divided by this = this / n

integer crypto::integer::operator<< (uint16_t n) const

Left shift operator.

Parameters

in	*n*	Number of bits to shift

Returns

this << n

integer crypto::integer::operator>> (uint16_t n) const

Right shift operator.

Parameters

in	*n*	Number of bits to shift

Returns

this >> n

bool crypto::integer::prime (uint16_t testVal = `algo::primeTestCycle`) const

Test if this integer is prime.

Preforms a probabilistic prime test on this number. This operation can be quite expensive, especially for large numbers.

Parameters

`in`	*testVal*	Number of test cycles, crytpo::algo::primeTestCycle by default

Returns

true if prime, else, false

static **integer** crypto::integer::two ( ) `[static]`

Constructs a '2' integer.

Returns

2

static **integer** crypto::integer::zero ( ) `[inline]`, `[static]`

Constructs a '0' integer.

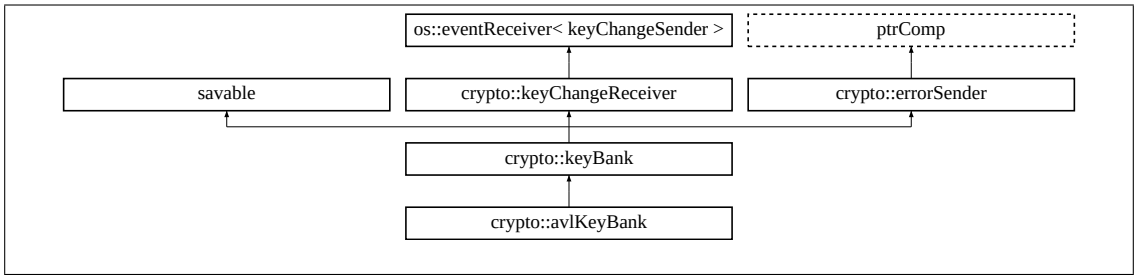
Returns

0

6.22 crypto::keyBank Class Reference

Key bank interface.

Inheritance diagram for crypto::keyBank:



Public Member Functions

- virtual ~**keyBank** ()

  Virtual destructor.

- virtual os::smart_ptr< **nodeGroup** > **addPair** (std::string groupName, std::string name, os↩ ::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)=0

  Adds authenticated node to bank.
- virtual void **save** ()=0

  Saves bank to file.
- const std::string & **savePath** () const

  Get save path.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeNameReference** > name)=0

  Find by group name reference.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **nodeKeyReference** > key)=0

  Find by group key reference.
- virtual os::smart_ptr< **nodeGroup** > **find** (std::string groupName, std::string name)

  Find by group name and name.
- virtual os::smart_ptr< **nodeGroup** > **find** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Find by key information.
- void **setPassword** (const unsigned char ∗key=NULL, unsigned int keyLen=0)

  Set password.
- void **setStreamPackage** (os::smart_ptr< **streamPackageFrame** > strmPack)

  Set stream package.
- void **setPublicKey** (os::smart_ptr< **publicKey** > pubKey)

  Set public key.

Protected Member Functions

- virtual void **pushNewNode** (os::smart_ptr< **nodeNameReference** > name)=0

  Add name node.
- virtual void **pushNewNode** (os::smart_ptr< **nodeKeyReference** > key)=0

  Add key node.
- virtual void **load** ()=0

  Loads bank from file.
- void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggers on key change.
- os::smart_ptr< **nodeGroup** > **fileLoadHelper** (os::smartXMLNode xmlTree)

  Construct node with XML tree.
- **keyBank** (std::string **savePath**, const unsigned char ∗key=NULL, unsigned int keyLen=0, os↩ ::smart_ptr< **streamPackageFrame** > strmPck=NULL)

  Construct with save path and key.
- **keyBank** (std::string **savePath**, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **stream**↩ **PackageFrame** > strmPck=NULL)

  Construct with save path and public key.

Protected Attributes

- os::smart_ptr< **streamPackageFrame** > **_streamPackage**

  Stream package.
- unsigned char * **_symKey**

  Primary symmetric key.
- unsigned int **_keyLen**

  Length of symmetric key.
- os::smart_ptr< **publicKey** > **_pubKey**

  Public key group to encrypt file.

Private Attributes

- std::string **_savePath**

  Path to save file.

Friends

- class **nodeGroup**

  Friendship with node grouping.

6.22.1 Detailed Description

Key bank interface.

Acts as an interface for classes which allow for the storing, saving and searching of cyptographic keys. These banks act, in essense, as data-bases.

6.22.2 Constructor & Destructor Documentation

crypto::keyBank::keyBank (std::string savePath, const unsigned char * key = `NULL`, unsigned int keyLen = `0`, os::smart_ptr< **streamPackageFrame** > strmPck = `NULL`) `[protected]`

Construct with save path and key.

Parameters

`in`	*savePath*	Path to save file
`in`	*key*	Symetric key
`in`	*keyLen*	Length of symetric key
`in`	*strmPck*	Definition of algorithms used

crypto::keyBank::keyBank (std::string savePath, os::smart_ptr< **publicKey** > pubKey, os::smart_ptr< **streamPackageFrame** > strmPck = `NULL`) `[protected]`

Construct with save path and public key.

Parameters

in	*savePath*	Path to save file
in	*pubKey*	Public key
in	*strmPck*	Definition of algorithms used

virtual crypto::keyBank::~keyBank ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.22.3 Member Function Documentation

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::addPair (std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [pure virtual]

Adds authenticated node to bank.

Note that if a node has not be authenticated, adding it to the bank will cause a potential security vulnerability. Nodes should be authenticated before being added to the bank.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node
in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Return reference to the new node group

Implemented in **crypto::avlKeyBank** (p. 62).

os::smart_ptr<**nodeGroup**> crypto::keyBank::fileLoadHelper (os::smartXMLNode xmlTree) [inline], [protected]

Construct node with XML tree.

Parameters

in	*xmlTree*	XML tree from file

Returns

Node group constructed with tree

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **nodeNameReference** > name) [pure virtual]

Find by group name reference.

Parameters

in	*name*	Name reference to be searched

Returns

Node group found by arguments

Implemented in **crypto::avlKeyBank** (p. 62).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **nodeKeyReference** > key) [pure virtual]

Find by group key reference.

Parameters

in	*key*	Key reference to be searched

Returns

Node group found by arguments

Implemented in **crypto::avlKeyBank** (p. 62).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (std::string groupName, std::string name) [inline], [virtual]

Find by group name and name.

Parameters

in	*groupName*	Name of the node's group
in	*name*	Name of the node

Returns

Node group found by arguments

Reimplemented in **crypto::avlKeyBank** (p. 63).

virtual os::smart_ptr<**nodeGroup**> crypto::keyBank::find (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) [inline], [virtual]

Find by key information.

Parameters

in	*key*	Key of node to be added
in	*algoID*	ID of algorithm for key
in	*keySize*	Length of key of the node

Returns

Node group found by arguments

Reimplemented in **crypto::avlKeyBank** (p. 63).

virtual void crypto::keyBank::load ( ) [protected], [pure virtual]

Loads bank from file.

Returns

void

Implemented in **crypto::avlKeyBank** (p. 63).

void crypto::keyBank::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) [protected], [virtual]

Triggers on key change.
Marks this class for re-saving when the public key has been re-generated.

Parameters

in	*pbk*	Public key which was changed

Returns

void

Reimplemented from **crypto::keyChangeReceiver** (p. 145).

virtual void crypto::keyBank::pushNewNode (os::smart_ptr< **nodeNameReference** > name) [protected], [pure virtual]

Add name node.
Inserts a name node into the bank. The name node has a reference to a node group.

Parameters

in	*name*	Name node to be added

Returns

void

Implemented in **crypto::avlKeyBank** (p. 64).

virtual void crypto::keyBank::pushNewNode (os::smart_ptr< **nodeKeyReference** > key) [protected], [pure virtual]

Add key node.

Inserts a key node into the bank. The key node has a reference to a node group.

Parameters

in	*key*	Key node to be added

Returns

void

Implemented in **crypto::avlKeyBank** (p. 64).

virtual void crypto::keyBank::save ( ) [pure virtual]

Saves bank to file.

Returns

void

Implemented in **crypto::avlKeyBank** (p. 64).

const std::string& crypto::keyBank::savePath ( ) const [inline]

Get save path.

Returns

crypto::keyBank::_savePath (p. 143)

void crypto::keyBank::setPassword (const unsigned char * key = NULL, unsigned int keyLen = 0)

Set password.

Sets symetric key used to securely save user data.

Parameters

in	*key*	Symetric key
in	*keyLen*	Length of symetric key

Returns

void

void crypto::keyBank::setPublicKey (os::smart_ptr< **publicKey** > pubKey)

Set public key.
Binds a new public key to this. Calls for saving of this user.

Parameters

in	*pubKey*	Public key

Returns

void

void crypto::keyBank::setStreamPackage (os::smart_ptr< **streamPackageFrame** > strmPack)

Set stream package.
Binds a new stream package. Calls for saving of this user.

Parameters

in	*strmPack*	Stream package

Returns

void

6.22.4 Friends And Related Function Documentation

friend class **nodeGroup** [friend]

Friendship with node grouping.
Node groups must be able to push name and key nodes onto the key bank.

6.22.5 Member Data Documentation

unsigned int crypto::keyBank::_keyLen [protected]

Length of symmetric key.

os::smart_ptr<**publicKey**> crypto::keyBank::_pubKey [protected]

Public key group to encrypt file.

std::string crypto::keyBank::_savePath [private]

Path to save file.

os::smart_ptr<**streamPackageFrame**> crypto::keyBank::_streamPackage [protected]

Stream package.

Used for the saving of the key bank. This defines the algorithms used for encrypting the saved bank, if it is encrypted.

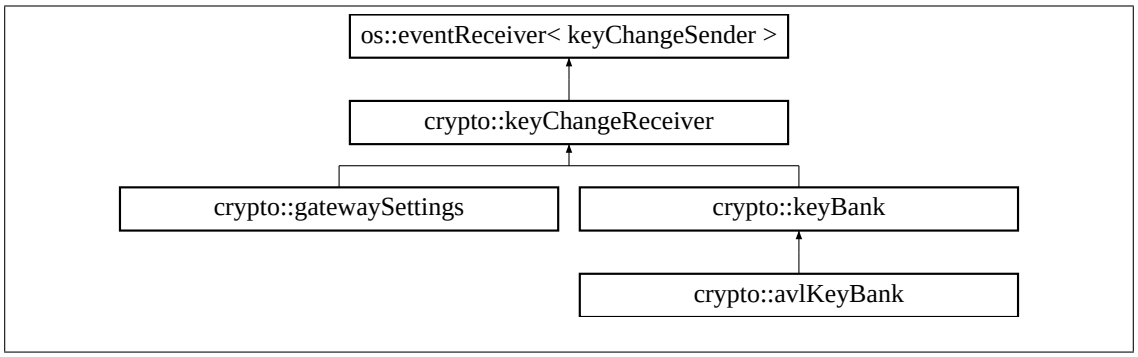
unsigned char* crypto::keyBank::_symKey [protected]

Primary symmetric key.

6.23 crypto::keyChangeReceiver Class Reference

Interface for receiving key changes.

Inheritance diagram for crypto::keyChangeReceiver:



Public Member Functions

- virtual ~**keyChangeReceiver** ()

  Virtual destructor.
- virtual bool **operator==** (const **keyChangeReceiver** &l) const

  Equality test.
- virtual bool **operator**> (const **keyChangeReceiver** &l) const

  Greater than test.
- virtual bool **operator**< (const **keyChangeReceiver** &l) const

  Less than test.
- virtual bool **operator**>**=** (const **keyChangeReceiver** &l) const

  Greater than/equal to test.
- virtual bool **operator**<**=** (const **keyChangeReceiver** &l) const

  Less than/equal to test.

Protected Member Functions

- virtual void **publicKeyChanged** (os::smart_ptr< **publicKey** > pbk)

  Triggers on key change.

Friends

- class **keyChangeSender**

  *Allows access to **crypto::keyChangeReceiver::publicKeyChanged** (p. 145).*

6.23.1 Detailed Description

Interface for receiving key changes.

A class which is alerted by public keys when the public key is updated.

6.23.2 Constructor & Destructor Documentation

virtual crypto::keyChangeReceiver::~keyChangeReceiver ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.23.3 Member Function Documentation

virtual bool crypto::keyChangeReceiver::operator< (const **keyChangeReceiver** & l) const `[inline]`, `[virtual]`

Less than test.

virtual bool crypto::keyChangeReceiver::operator<= (const **keyChangeReceiver** & l) const `[inline]`, `[virtual]`

Less than/equal to test.

virtual bool crypto::keyChangeReceiver::operator== (const **keyChangeReceiver** & l) const `[inline]`, `[virtual]`

Equality test.

virtual bool crypto::keyChangeReceiver::operator> (const **keyChangeReceiver** & l) const `[inline]`, `[virtual]`

Greater than test.

virtual bool crypto::keyChangeReceiver::operator>= (const **keyChangeReceiver** & l) const `[inline]`, `[virtual]`

Greater than/equal to test.

virtual void crypto::keyChangeReceiver::publicKeyChanged (os::smart_ptr< **publicKey** > pbk) `[inline]`, `[protected]`, `[virtual]`

Triggers on key change.

Is triggered by **crypto::publicKey** (p. 199) whenever the public key is updated.

Parameters

in	*pbk*	Public key which was changed

Returns

void

Reimplemented in **crypto::keyBank** (p. 141), and **crypto::gatewaySettings** (p. 114).

6.23.4 Friends And Related Function Documentation

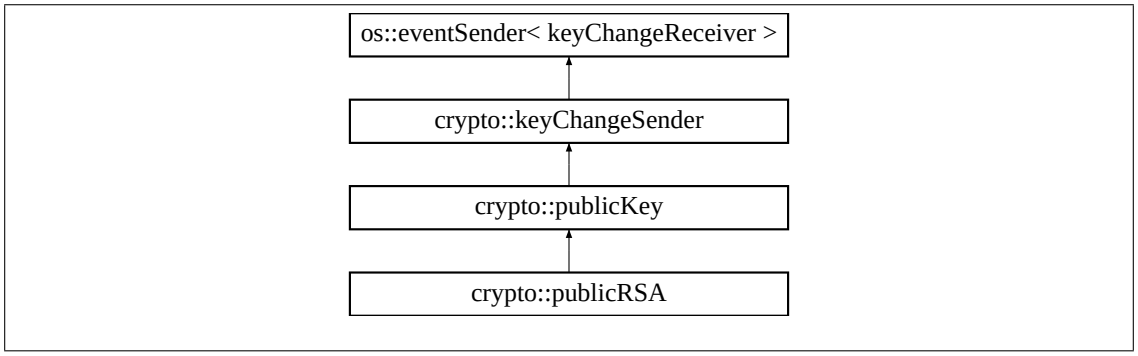
friend class **keyChangeSender** [friend]

Allows access to **crypto::keyChangeReceiver::publicKeyChanged** (p. 145).

6.24 crypto::keyChangeSender Class Reference

Interface inherited by **publicKey** (p. 199).

Inheritance diagram for crypto::keyChangeSender:



Public Member Functions

- virtual ~**keyChangeSender** ()
  Virtual destructor.
- virtual bool **operator==** (const **keyChangeSender** &I) const
  Equality test.
- virtual bool **operator**> (const **keyChangeSender** &I) const
  Greater than test.
- virtual bool **operator**< (const **keyChangeSender** &I) const
  Less than test.
- virtual bool **operator**>**=** (const **keyChangeSender** &I) const
  Greater than/equal to test.
- virtual bool **operator**<**=** (const **keyChangeSender** &I) const
  Less than/equal to test.

Protected Member Functions

- void **sendEvent** (os::smart_ptr< **keyChangeReceiver** > ptr)

  Sends key change event to listeners.

6.24.1 Detailed Description

Interface inherited by **publicKey** (p. 199).

This class is meaningless outside of **crypto::publicKey** (p. 199) and is only designed to be inherited by **publicKey** (p. 199) to interface with **crypto::keyChangeReceiver** (p. 144).

6.24.2 Constructor & Destructor Documentation

virtual crypto::keyChangeSender::~keyChangeSender ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.24.3 Member Function Documentation

virtual bool crypto::keyChangeSender::operator< (const **keyChangeSender** & l) const [inline], [virtual]

Less than test.

virtual bool crypto::keyChangeSender::operator<= (const **keyChangeSender** & l) const [inline], [virtual]

Less than/equal to test.

virtual bool crypto::keyChangeSender::operator== (const **keyChangeSender** & l) const [inline], [virtual]

Equality test.

virtual bool crypto::keyChangeSender::operator> (const **keyChangeSender** & l) const [inline], [virtual]

Greater than test.

virtual bool crypto::keyChangeSender::operator>= (const **keyChangeSender** & l) const [inline], [virtual]

Greater than/equal to test.

void crypto::keyChangeSender::sendEvent (os::smart_ptr< **keyChangeReceiver** > ptr)
[inline], [protected]

Sends key change event to listeners.

Useing the interface provided by the os::eventSender class, alert any classes listening for a public key change that one has occured.

Parameters

in	*ptr*	Receiver to alert

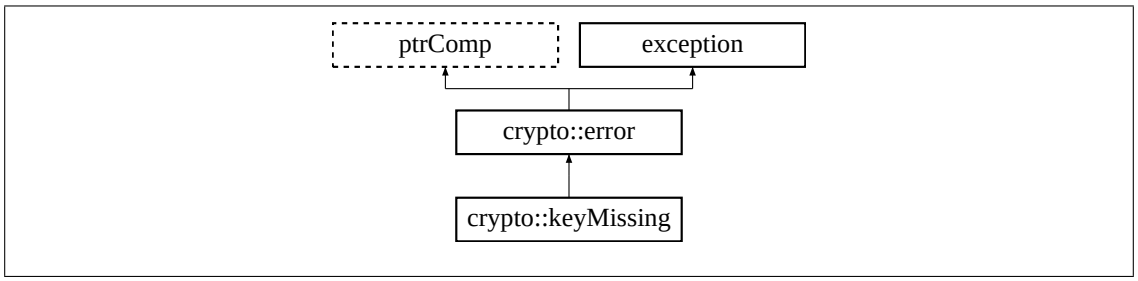
Returns

void

6.25 crypto::keyMissing Class Reference

Key missing error.

Inheritance diagram for crypto::keyMissing:



Public Member Functions

- virtual ~**keyMissing** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Key missing".
- std::string **errorDescription** () const

  Long error descriptor Returns "Cannot decrypt the data stream, the key is missing!".

6.25.1 Detailed Description

Key missing error.

Thrown when a key cannot be found to decrypt the incoming data stream

6.25.2 Constructor & Destructor Documentation

virtual crypto::keyMissing::~keyMissing ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.25.3 Member Function Documentation

std::string crypto::keyMissing::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Cannot decrypt the data stream, the key is missing!".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::keyMissing::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Key missing".

Returns

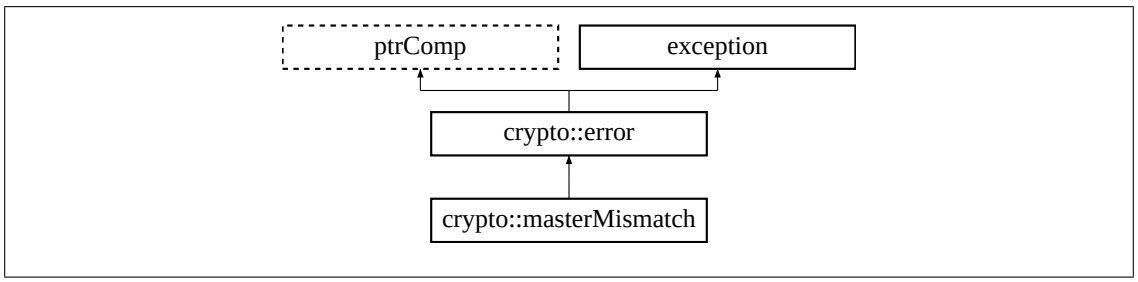
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.26 crypto::masterMismatch Class Reference

Master mis-match.

Inheritance diagram for crypto::masterMismatch:



Public Member Functions

- virtual ~**masterMismatch** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Master Comparison Mis-match".
- std::string **errorDescription** () const

  Long error descriptor Returns "Two nodes which are interacting have different masters!".

6.26.1 Detailed Description

Master mis-match.

Thrown when two elements attempt an interaction but have different masters.

6.26.2 Constructor & Destructor Documentation

virtual crypto::masterMismatch::~masterMismatch ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.26.3 Member Function Documentation

std::string crypto::masterMismatch::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Two nodes which are interacting have different masters!".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::masterMismatch::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Master Comparison Mis-match".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.27 crypto::message Class Reference

Crypto-Gateway message.

Public Member Functions

- **message** (uint16_t sz)

  Constructs message with a size.
- **message** (const **message** &msg)

  Copy constructor.
- virtual ~**message** ()

  Virtual destructor.
- uint16_t **messageSize** () const

  Return message size.
- uint16_t **size** () const

Return message packet size.

- uint16_t **encryptionDepth** () const

  Return level of message encryption.
- uint8_t * **data** ()

  Modifiable data pointer.
- const uint8_t * **data** () const

  Immutable data pointer.
- bool **encrypted** () const

  Is the message encrypted.
- bool **pushString** (std::string s)

  Add string to this message.
- std::string **popString** ()

  Remove string from this message.

Static Public Member Functions

- static **message encryptedMessage** (uint8_t *rawData, uint16_t sz)

  Constructs an encrypted message.
- static **message decryptedMessage** (uint8_t *rawData, uint16_t sz)

  Constructs an decrypted message.

Static Public Attributes

- static const uint8_t **BLOCKED** =0

  Blocked message tag.
- static const uint8_t **PING** =1

  Ping message tag.
- static const uint8_t **FORWARD** =2

  Forward message tag.
- static const uint8_t **STREAM_KEY** =3

  Stream key message tag.
- static const uint8_t **SIGNING_MESSAGE** =4

  Signing message tag.
- static const uint8_t **SECURE_DATA_EXCHANGE** =5

  Secure data exchange message tag.
- static const uint8_t **CONFIRM_ERROR** =252

  Confirm error message tag.
- static const uint8_t **BASIC_ERROR** =253

  Basic error message tag.
- static const uint8_t **TIMEOUT_ERROR** =254

  Timeout error message tag.
- static const uint8_t **PERMENANT_ERROR** =255

  Permenant error message tag.

Private Attributes

- uint16_t **_messageSize**

  Size of message.
- uint16_t **_size**

  Size of the message packet.
- uint16_t **_encryptionDepth**

  Depth of encryption.
- uint8_t * **_data**

  Data in the message packet.

Friends

- class **gatewaySettings**

  Friendship with settings.
- class **gateway**

  Friendship with gateway.

6.27.1 Detailed Description

Crypto-Gateway message.

This message is meant to be passed between machines. The gateway either encrypts or decrypts the message. This message allows for nested encryption.

6.27.2 Constructor & Destructor Documentation

crypto::message::message (uint16_t sz)

Constructs message with a size.

Parameters

in	*sz*	Size of message

crypto::message::message (const **message** & msg)

Copy constructor.

Parameters

in	*msg*	Message to be copied

virtual crypto::message::~message ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.27.3 Member Function Documentation

uint8_t* crypto::message::data ( ) `[inline]`

Modifiable data pointer.

Returns

message::_data (p. 155)

const uint8_t* crypto::message::data ( ) const `[inline]`

Immutable data pointer.

Returns

message::_data (p. 155)

static **message** crypto::message::decryptedMessage (uint8_t * rawData, uint16_t sz) `[static]`

Constructs an decrypted message.

Parses an array of data assuming that the data in question has been generated outside of a gateway

Parameters

`in`	*rawData*	Incoming data array
`in`	*sz*	Size of incoming data

Returns

New message

bool crypto::message::encrypted ( ) const `[inline]`

Is the message encrypted.

Returns

True if encrypted, else, false

static **message** crypto::message::encryptedMessage (uint8_t * rawData, uint16_t sz) `[static]`

Constructs an encrypted message.

Parses an array of data assuming that the data in question has come out of another gateway.

Parameters

in	*rawData*	Incoming data array
in	*sz*	Size of incoming data

Returns

New message

uint16_t crypto::message::encryptionDepth ( ) const [inline]

Return level of message encryption.

Returns

message::_encryptionDepth (p. 155)

uint16_t crypto::message::messageSize ( ) const [inline]

Return message size.

Returns

message::_messageSize (p. 155)

std::string crypto::message::popString ( )

Remove string from this message.

Returns

Next string to remove

bool crypto::message::pushString (std::string s)

Add string to this message.

Returns

True if successful

uint16_t crypto::message::size ( ) const [inline]

Return message packet size.

Returns

message::_size (p. 155)

6.27.4 Friends And Related Function Documentation

friend class **gateway** [friend]

Friendship with gateway.

The **crypto::gateway** (p. 92) class encrypts and decrypts messages, so it must be able to access the intrisics of the message.

friend class **gatewaySettings** [friend]

Friendship with settings.

The **crypto::gatewaySettings** (p. 107) class must be able to access the intrinsics of this class inorder to create and parse a ping message.

6.27.5 Member Data Documentation

uint8_t* crypto::message::_data [private]

Data in the message packet.

uint16_t crypto::message::_encryptionDepth [private]

Depth of encryption.

Holds how many times this particular message has been encrypted.

uint16_t crypto::message::_messageSize [private]

Size of message.

This size refers to the size of the non-header and non-checksum bytes in the message. This value remains constant as messages are encrypted and decrypted.

uint16_t crypto::message::_size [private]

Size of the message packet.

This size includes all support data along with the meaningful message.

const uint8_t crypto::message::BASIC_ERROR =253 [static]

Basic error message tag.

Sent by a gateway when a basic error occurs.

const uint8_t crypto::message::BLOCKED =0 [static]

Blocked message tag.

Indicates that the node sending the particular message has blocked the node receiving the particular message.

const uint8_t crypto::message::CONFIRM_ERROR =252 [static]

Confirm error message tag.

Messages of this type are sent to allow the receiving gateway to know that the sending gateway has acknowledged its error.

const uint8_t crypto::message::FORWARD =2 [static]

Forward message tag.

Indicates a message is being sent through this gateway to another gateway for final decryption.

const uint8_t crypto::message::PERMENANT_ERROR =255 [static]

Permenant error message tag.

Sent by a gateway when a permenant error has occurred. Permenant errors never expire, and a gateway will never reconnect once a permenant error has occurred.

const uint8_t crypto::message::PING =1 [static]

Ping message tag.

Message type sent by gateways when exchanging names and public keys.

const uint8_t crypto::message::SECURE_DATA_EXCHANGE =5 [static]

Secure data exchange message tag.

Message passed between two gateways when secure. Used by the gateways to notify connected gateways when keys and algorithms change after a connection has been secured.

const uint8_t crypto::message::SIGNING_MESSAGE =4 [static]

Signing message tag.

Indicates a message is cryptographically establishing the identity of a node.

const uint8_t crypto::message::STREAM_KEY =3 [static]

Stream key message tag.

Indicates a message is exchanging stream cipher keys through the defined public key algorithm.

const uint8_t crypto::message::TIMEOUT_ERROR =254 [static]

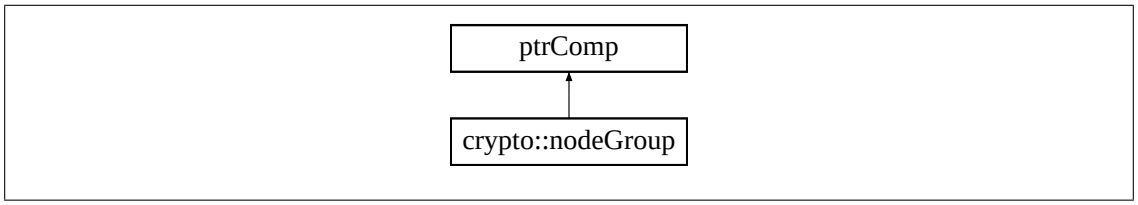
Timeout error message tag.

Sent by a gateway when a timeout error occurs. Timeout errors are more serious and take a certain amount of time to expire.

6.28 crypto::nodeGroup Class Reference

Node group.

Inheritance diagram for crypto::nodeGroup:



Public Member Functions

- **nodeGroup** (**keyBank** *master, std::string groupName, std::string **name**, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

  Node group constructor.
- virtual ~**nodeGroup** ()

  Virtual destructor.
- void **getName** (std::string &groupName, std::string &**name**)

  Allows access to the most recent name.
- std::string **name** ()

  Concatenated name.
- os::smart_ptr< os::adnode< **nodeNameReference** > > **getFirstName** ()

  Returns first name in the list.
- os::smart_ptr< os::adnode< **nodeKeyReference** > > **getFirstKey** ()

  Returns first key in the list.
- void **merge** (**nodeGroup** &source)

  Merge a node group into this.
- void **addAlias** (std::string groupName, std::string **name**, uint64_t timestamp=os::getTimestamp())

  Add new alias for group.
- void **addKey** (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp=os↩ ::getTimestamp())

  Add new key for group.
- unsigned int **numberOfNames** () const

  Returns the number of names.
- unsigned int **numberOfKeys** () const

  Returns the number of keys.
- os::smart_ptr< os::smart_ptr< **nodeNameReference** > > **namesByTimestamp** (unsigned int &size)

  Returns names sorted by timestamp.
- os::smart_ptr< os::smart_ptr< **nodeKeyReference** > > **keysByTimestamp** (unsigned int &size)

  Returns keys sorted by timestamp.
- os::smartXMLNode **buildXML** ()

  Build XML tree.

Private Member Functions

- void **sortKeys** ()

  Sorts keys by timestamp.
- void **sortNames** ()

  Sorts names by timestamp.
- **nodeGroup** (**keyBank** *master, os::smartXMLNode fileNode)

  Node group constructor.

Private Attributes

- **keyBank** * **_master**

  Pointer to key bank.
- os::asyncAVLTree< **nodeNameReference** > **nameList**

  List of all names associated with this node.
- os::asyncAVLTree< **nodeKeyReference** > **keyList**

  List of all keys associated with this node.
- std::mutex **sortingLock**

  Lock used for sorting.
- os::smart_ptr< os::smart_ptr< **nodeNameReference** > > **sortedNames**

  Array of names sorted by timestamp.
- os::smart_ptr< os::smart_ptr< **nodeKeyReference** > > **sortedKeys**

  Array of keys sorted by timestamp.

Friends

- class **keyBank**

  *Only **keyBank** (*p. *136) can load a node group.*

6.28.1 Detailed Description

Node group.

A list of all names and keys which are associated with a single node. This must exist because nodes can change their name during operation.

6.28.2 Constructor & Destructor Documentation

crypto::nodeGroup::nodeGroup (**keyBank** * master, os::smartXMLNode fileNode) `[private]`

Node group constructor.

Constructs a node group with an XML tree. This constructor throws exceptions if errors occur.

Parameters

	[in/out]	master Reference to the 'master' group holder
`in`	*fileNode*	XML root which defines the group

crypto::nodeGroup::nodeGroup (**keyBank** * master, std::string groupName, std::string name, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize)

Node group constructor.

Parameters

	[in/out]	master Reference to the 'master' group holder

Parameters

in	*groupName*	Group name of the node being registered
in	*name*	Name of the node being registered
in	*key*	The public key of a given node
in	*algoID*	The algorithm identifier
in	*keySize*	Size of the key provided

virtual crypto::nodeGroup::~nodeGroup ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.28.3 Member Function Documentation

void crypto::nodeGroup::addAlias (std::string groupName, std::string name, uint64_t timestamp = os::getTimestamp())

Add new alias for group.

Parameters

in	*groupName*	Group name of the node being registered
in	*name*	Name of the node being registered
	timestamp	The time this node was created, 'now' by defult

Returns

void

void crypto::nodeGroup::addKey (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp = os::getTimestamp())

Add new key for group.

Parameters

in	*key*	The public key of a given node
in	*algoID*	The algorithm identifier
in	*keySize*	Size of the key provided
	timestamp	The time this node was created, 'now' by defult

Returns

void

os::smartXMLNode crypto::nodeGroup::buildXML ( )

Build XML tree.

Builds an XML tree from this node group. This tree is designed to be saved by the key bank.

Returns

Root of tree to be saved

os::smart_ptr<os::adnode<**nodeKeyReference**> > crypto::nodeGroup::getFirstKey ( ) `[inline]`

Returns first key in the list.

This function returns an alphabetical order. Note that it is often the case that a user needs to sort by timestamp. This functionality is also provided.

Returns

crypto::nodeGroup::keyList.getFirst()

os::smart_ptr<os::adnode<**nodeNameReference**> > crypto::nodeGroup::getFirstName ( ) `[inline]`

Returns first name in the list.

This function returns an alphabetical order. Note that it is often the case that a user needs to sort by timestamp. This functionality is also provided.

Returns

crypto::nodeGroup::nameList.getFirst()

void crypto::nodeGroup::getName (std::string & groupName, std::string & name)

Allows access to the most recent name.

Parameters

`out`	*groupName*	**crypto::nodeGroup::sortedNames** (p. 162)[0]->groupName()
`out`	*name*	**crypto::nodeGroup::sortedNames** (p. 162)[0]->**name()** (p. 161)

Returns

void

os::smart_ptr<os::smart_ptr<**nodeKeyReference**> > crypto::nodeGroup::keysByTimestamp (unsigned int & size)

Returns keys sorted by timestamp.

Parameters

out	*size*	Size of array to be returned

Returns

crypto::nodeGroup::sortedKeys (p. 162)

void crypto::nodeGroup::merge (**nodeGroup** & source)

Merge a node group into this.

Acheives merge entirely by reference. It is assumed that the node being merged into this node will shortly be deleted.

Parameters

in	*source*	Node group to merge

Returns

void

std::string crypto::nodeGroup::name ( )

Concatenated name.

Concatenated the groupName and name and then returns the combination.
return groupName+":"+name

os::smart_ptr<os::smart_ptr<**nodeNameReference**> > crypto::nodeGroup::namesByTimestamp (unsigned int & size)

Returns names sorted by timestamp.

Parameters

out	*size*	Size of array to be returned

Returns

crypto::nodeGroup::sortedNames (p. 162)

unsigned int crypto::nodeGroup::numberOfKeys ( ) const `[inline]`

Returns the number of keys.

Returns

crypto::nodeGroup::keyList.size()

unsigned int crypto::nodeGroup::numberOfNames ( ) const [inline]

Returns the number of names.

Returns

crypto::nodeGroup::nameList.size()

void crypto::nodeGroup::sortKeys ( ) [private]

Sorts keys by timestamp.

void crypto::nodeGroup::sortNames ( ) [private]

Sorts names by timestamp.

6.28.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Only **keyBank** (p. 136) can load a node group.

6.28.5 Member Data Documentation

keyBank* crypto::nodeGroup::_master [private]

Pointer to key bank.

os::asyncAVLTree<**nodeKeyReference**> crypto::nodeGroup::keyList [private]

List of all keys associated with this node.

os::asyncAVLTree<**nodeNameReference**> crypto::nodeGroup::nameList [private]

List of all names associated with this node.

os::smart_ptr<os::smart_ptr<**nodeKeyReference**> > crypto::nodeGroup::sortedKeys [private]

Array of keys sorted by timestamp.

os::smart_ptr<os::smart_ptr<**nodeNameReference**> > crypto::nodeGroup::sortedNames [private]

Array of names sorted by timestamp.

std::mutex crypto::nodeGroup::sortingLock [private]

Lock used for sorting.

6.29 crypto::nodeKeyReference Class Reference

Key storage node.

Public Member Functions

- virtual ~**nodeKeyReference** ()

  Virtual destructor.
- **nodeGroup** ∗ **master** ()

  Returns a pointer to its master.
- os::smart_ptr< **number** > **key** () const

  Returns the key.
- uint16_t **algoID** () const

  Returns the algorithm key.
- uint16_t **keySize** () const

  Returns the key size.
- uint64_t **timestamp** () const

  Returns the timestamp.
- int **compare** (const **nodeKeyReference** &comp) const

  *Compare **crypto::nodeKeyReference*** *(p. 163).*
- bool **operator==** (const **nodeKeyReference** &comp) const

  Equality operator.
- bool **operator!=** (const **nodeKeyReference** &comp) const

  Not-equals operator.
- bool **operator**> (const **nodeKeyReference** &comp) const

  Greater-than operator.
- bool **operator**>**=** (const **nodeKeyReference** &comp) const

  Greater-than/equals to operator.
- bool **operator**< (const **nodeKeyReference** &comp) const

  Less-than operator.
- bool **operator**<**=** (const **nodeKeyReference** &comp) const

  Less-than/equals to operator.

Private Member Functions

- **nodeKeyReference** (**nodeGroup** ∗**master**, os::smart_ptr< **number** > **key**, uint16_t **algoID**, uint16_t **keySize**, uint64_t **timestamp**=os::getTimestamp())

  Key reference node constructor.
- **nodeKeyReference** (os::smart_ptr< **number** > **key**, uint16_t **algoID**, uint16_t **keySize**)

  Key reference node constructor for searching.

Private Attributes

- **nodeGroup** * **_master**

  Pointer to node group.
- os::smart_ptr< **number** > **_key**

  Shared pointer to public key.
- uint16_t **_algoID**

  ID of public key algorithm.
- uint16_t **_keySize**

  Size of public key.
- uint64_t **_timestamp**

  Timestamp key created.

Friends

- class **nodeGroup**

  *Friendship with **crypto::nodeGroup** (p. 156).*
- class **keyBank**

  *Friendship with **crypto::keyBank** (p. 136).*

6.29.1 Detailed Description

Key storage node.

Allows for storage and sorting of a node group by its key. This node holds a reference to the larger group node.

6.29.2 Constructor & Destructor Documentation

crypto::nodeKeyReference::nodeKeyReference (**nodeGroup** * master, os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize, uint64_t timestamp = `os::getTimestamp()`) `[private]`

Key reference node constructor.

Parameters

	[in/out]	master Reference to the 'master' group
`in`	*key*	The public key of a given node
`in`	*algoID*	The algorithm identifier
`in`	*keySize*	Size of the key provided
	timestamp	The time this node was created, 'now' by defult

crypto::nodeKeyReference::nodeKeyReference (os::smart_ptr< **number** > key, uint16_t algoID, uint16_t keySize) `[private]`

Key reference node constructor for searching.

Parameters

in	*key*	The public key of a given node
in	*algoID*	The algorithm identifier
in	*keySize*	Size of the key provided

virtual crypto::nodeKeyReference::~nodeKeyReference ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.29.3 Member Function Documentation

uint16_t crypto::nodeKeyReference::algoID ( ) const [inline]

Returns the algorithm key.

Returns

crypto::nodeKeyReference::_algoID (p. 168)

int crypto::nodeKeyReference::compare (const **nodeKeyReference** & comp) const

Compare **crypto::nodeKeyReference** (p. 163).

Compares two node key references by their public key, returning the result in the form of a 1,0 or -1.

Parameters

in	*comp*	Key reference to compare against

Returns

1, 0, -1 (Greater than, equal to, less than)

os::smart_ptr<**number**> crypto::nodeKeyReference::key ( ) const [inline]

Returns the key.

Returns

crypto::nodeKeyReference::_key (p. 168)

uint16_t crypto::nodeKeyReference::keySize ( ) const [inline]

Returns the key size.

Returns

crypto::nodeKeyReference::_keySize (p. 168)

nodeGroup* crypto::nodeKeyReference::master ( ) `[inline]`

Returns a pointer to its master.

Returns

crypto::~nodeKeyReference::_master

bool crypto::nodeKeyReference::operator!= (const **nodeKeyReference** & comp) const `[inline]`

Not-equals operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if not equal, else, false

bool crypto::nodeKeyReference::operator< (const **nodeKeyReference** & comp) const `[inline]`

Less-than operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if less than, else, false

bool crypto::nodeKeyReference::operator<= (const **nodeKeyReference** & comp) const `[inline]`

Less-than/equals to operator.

Parameters

in	*comp*	Key reference to compare against

Returns

true if less than or equal to, else, false

bool crypto::nodeKeyReference::operator== (const **nodeKeyReference** & comp) const `[inline]`

Equality operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if equal, else, false

bool crypto::nodeKeyReference::operator> (const **nodeKeyReference** & comp) const `[inline]`

Greater-than operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if greater than, else, false

bool crypto::nodeKeyReference::operator>= (const **nodeKeyReference** & comp) const `[inline]`

Greater-than/equals to operator.

Parameters

`in`	*comp*	Key reference to compare against

Returns

true if greater than or equal to, else, false

uint64_t crypto::nodeKeyReference::timestamp ( ) const `[inline]`

Returns the timestamp.

Returns

crypto::nodeKeyReference::_timestamp (p. 168)

6.29.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Friendship with **crypto::keyBank** (p. 136).
The key bank must be able to create a node key to search by key

friend class **nodeGroup** [friend]

Friendship with **crypto::nodeGroup** (p. 156).
Only node groupings can meaningfully create this class, so the constructor is private and only accessable by **crypto::nodeGroup** (p. 156).

6.29.5 Member Data Documentation

uint16_t crypto::nodeKeyReference::_algoID [private]

ID of public key algorithm.

os::smart_ptr<**number**> crypto::nodeKeyReference::_key [private]

Shared pointer to public key.

uint16_t crypto::nodeKeyReference::_keySize [private]

Size of public key.

nodeGroup∗ crypto::nodeKeyReference::_master [private]

Pointer to node group.

uint64_t crypto::nodeKeyReference::_timestamp [private]

Timestamp key created.

6.30 crypto::nodeNameReference Class Reference

Name storage node.

Public Member Functions

- virtual ~**nodeNameReference** ()
  Virtual destructor.
- **nodeGroup** ∗ **master** ()
  Returns a pointer to its master.
- std::string **groupName** () const
  Returns the group name.
- std::string **name** () const
  Returns the name.
- uint64_t **timestamp** () const

Returns the timestamp.

- int **compare** (const **nodeNameReference** &comp) const

  *Compare **crypto::nodeNameReference** (p. 168).*
- bool **operator==** (const **nodeNameReference** &comp) const

  Equality operator.
- bool **operator!=** (const **nodeNameReference** &comp) const

  Not-equals operator.
- bool **operator**> (const **nodeNameReference** &comp) const

  Greater-than operator.
- bool **operator**>**=** (const **nodeNameReference** &comp) const

  Greater-than/equals to operator.
- bool **operator**< (const **nodeNameReference** &comp) const

  Less-than operator.
- bool **operator**<**=** (const **nodeNameReference** &comp) const

  Less-than/equals to operator.

Private Member Functions

- **nodeNameReference** (**nodeGroup** ∗**master**, std::string **groupName**, std::string **name**, uint64↩ _t **timestamp**=os::getTimestamp())

  Name reference node constructor.
- **nodeNameReference** (std::string **groupName**, std::string **name**)

  Name reference node constructor for searching.

Private Attributes

- **nodeGroup** ∗ **_master**

  Pointer to node group.
- std::string **_groupName**

  Name of the group this name is from.
- std::string **_name**

  Name of the node.
- uint64_t **_timestamp**

  Timestamp key created.

Friends

- class **nodeGroup**

  *Friendship with **crypto::nodeGroup** (p. 156).*
- class **keyBank**

  *Friendship with **crypto::keyBank** (p. 136).*

6.30.1 Detailed Description

Name storage node.

Allows for storage and sorting of a node group by its name. This node holds a reference to the larger group node.

6.30.2 Constructor & Destructor Documentation

crypto::nodeNameReference::nodeNameReference (**nodeGroup** * master, std::string groupName, std::string name, uint64_t timestamp = `os::getTimestamp()`) `[private]`

Name reference node constructor.

Parameters

	[in/out]	master Reference to the 'master' group
`in`	*groupName*	Group name of the node being registered
`in`	*name*	Name of the node being registered
	timestamp	The time this node was created, 'now' by defult

crypto::nodeNameReference::nodeNameReference (std::string groupName, std::string name) `[private]`

Name reference node constructor for searching.

Parameters

`in`	*groupName*	Group name of the node being registered
`in`	*name*	Name of the node being registered

virtual crypto::nodeNameReference::~nodeNameReference ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.30.3 Member Function Documentation

int crypto::nodeNameReference::compare (const **nodeNameReference** & comp) const

Compare **crypto::nodeNameReference** (p. 168).

Compares two node name references by their group and name, returning the result in the form of a 1,0 or -1.

Parameters

`in`	*comp*	Name reference to compare against

Returns

1, 0, -1 (Greater than, equal to, less than)

std::string crypto::nodeNameReference::groupName ( ) const [inline]

Returns the group name.

Returns

crypto::nodeNameReference::_groupName (p. 173)

nodeGroup* crypto::nodeNameReference::master ( ) [inline]

Returns a pointer to its master.

Returns

crypto::nodeNameReference::_master (p. 173)

std::string crypto::nodeNameReference::name ( ) const [inline]

Returns the name.

Returns

crypto::nodeNameReference::_name (p. 173)

bool crypto::nodeNameReference::operator!= (const **nodeNameReference** & comp) const [inline]

Not-equals operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if not equal, else, false

bool crypto::nodeNameReference::operator< (const **nodeNameReference** & comp) const [inline]

Less-than operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if less than, else, false

bool crypto::nodeNameReference::operator<= (const **nodeNameReference** & comp) const `[inline]`

Less-than/equals to operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if less than or equal to, else, false

bool crypto::nodeNameReference::operator== (const **nodeNameReference** & comp) const `[inline]`

Equality operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if equal, else, false

bool crypto::nodeNameReference::operator> (const **nodeNameReference** & comp) const `[inline]`

Greater-than operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if greater than, else, false

bool crypto::nodeNameReference::operator>= (const **nodeNameReference** & comp) const `[inline]`

Greater-than/equals to operator.

Parameters

in	*comp*	Name reference to compare against

Returns

true if greater than or equal to, else, false

uint64_t crypto::nodeNameReference::timestamp ( ) const [inline]

Returns the timestamp.

Returns

crypto::nodeNameReference::_timestamp (p. 173)

6.30.4 Friends And Related Function Documentation

friend class **keyBank** [friend]

Friendship with **crypto::keyBank** (p. 136).

The key bank must be able to create a node name to search by name

friend class **nodeGroup** [friend]

Friendship with **crypto::nodeGroup** (p. 156).

Only node groupings can meaningfully create this class, so the constructor is private and only accessable by **crypto::nodeGroup** (p. 156).

6.30.5 Member Data Documentation

std::string crypto::nodeNameReference::_groupName [private]

Name of the group this name is from.

nodeGroup* crypto::nodeNameReference::_master [private]

Pointer to node group.

std::string crypto::nodeNameReference::_name [private]

Name of the node.

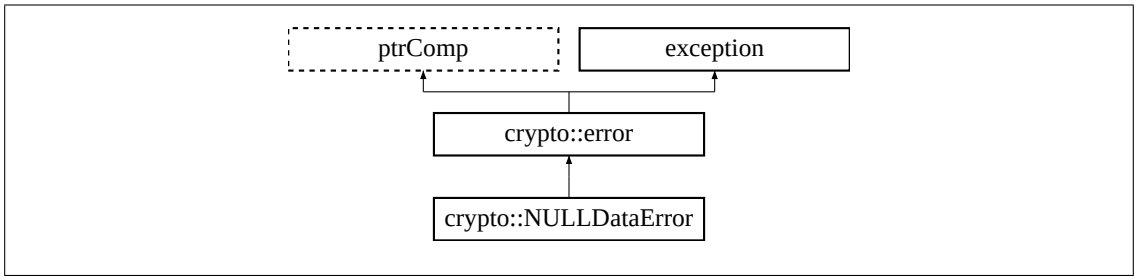
uint64_t crypto::nodeNameReference::_timestamp [private]

Timestamp key created.

6.31 crypto::NULLDataError Class Reference

NULL data error.

Inheritance diagram for crypto::NULLDataError:



Public Member Functions

- virtual ~**NULLDataError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "NULL Data".
- std::string **errorDescription** () const

  Long error descriptor Returns "A function was passed NULL data where this is illegal".

6.31.1 Detailed Description

NULL data error.

Thrown when NULL data is passed to a function or class.

6.31.2 Constructor & Destructor Documentation

virtual crypto::NULLDataError::~NULLDataError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.31.3 Member Function Documentation

std::string crypto::NULLDataError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "A function was passed NULL data where this is illegal".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::NULLDataError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "NULL Data".

Returns

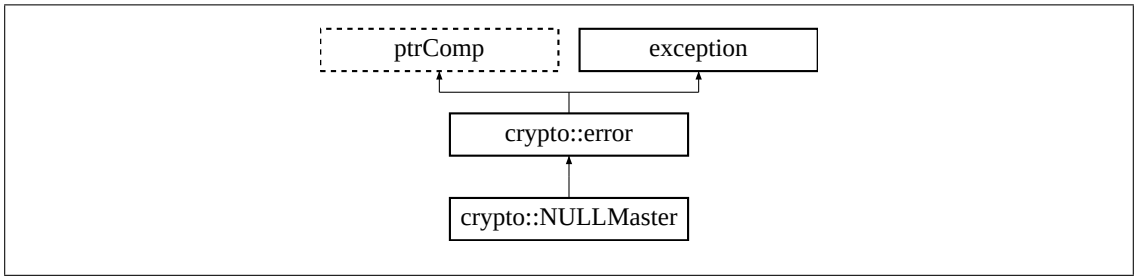
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.32 crypto::NULLMaster Class Reference

NULL master error.

Inheritance diagram for crypto::NULLMaster:



Public Member Functions

- virtual ~**NULLMaster** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "NULL Master pointer".
- std::string **errorDescription** () const

  Long error descriptor Returns "A class received a NULL master pointer, this is illegal".

6.32.1 Detailed Description

NULL master error.

Thrown when a class is passed a NULL master where such a class must have a defined master.

6.32.2 Constructor & Destructor Documentation

virtual crypto::NULLMaster::~NULLMaster ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.32.3 Member Function Documentation

std::string crypto::NULLMaster::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "A class received a NULL master pointer, this is illegal".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::NULLMaster::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "NULL Master pointer".

Returns

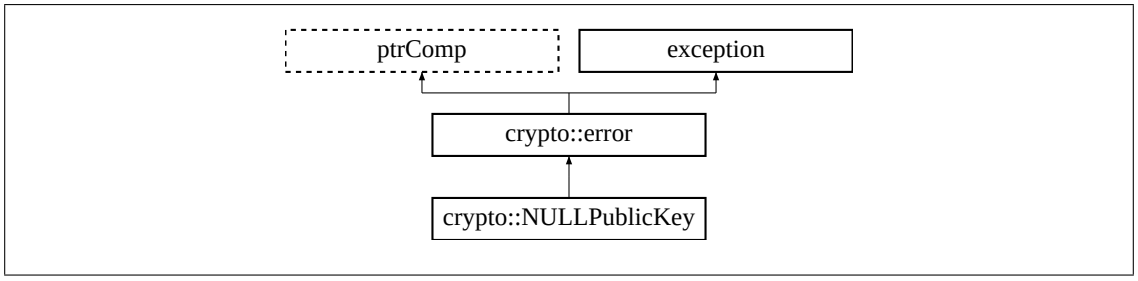
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.33 crypto::NULLPublicKey Class Reference

NULL public-key error.

Inheritance diagram for crypto::NULLPublicKey:



Public Member Functions

- virtual ~**NULLPublicKey** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Public Key NULL".
- std::string **errorDescription** () const

  Long error descriptor Returns "Attempted to bind a public key of illegal type NULL".

6.33.1 Detailed Description

NULL public-key error.

Thrown when a NULL public-key or public-key of undefined type is used.

6.33.2 Constructor & Destructor Documentation

virtual crypto::NULLPublicKey::~NULLPublicKey ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.33.3 Member Function Documentation

std::string crypto::NULLPublicKey::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Attempted to bind a public key of illegal type NULL".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::NULLPublicKey::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Public Key NULL".

Returns

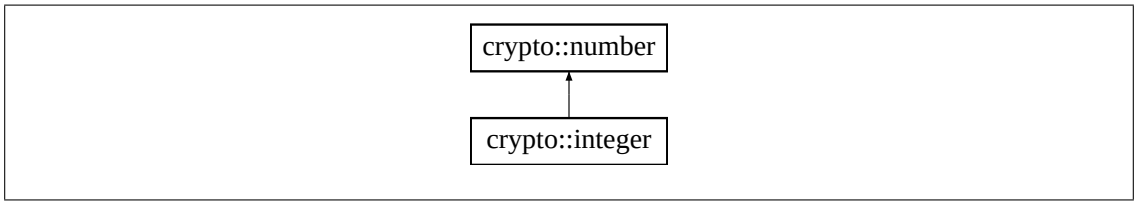
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.34 crypto::number Class Reference

Basic number definition.

Inheritance diagram for crypto::number:



Public Member Functions

- **number** (struct **numberType** *numDef=**buildNullNumberType**())

  Construct with number definition.
- **number** (uint16_t **size**, struct **numberType** *numDef=**buildNullNumberType**())

  Construct with size.
- **number** (const uint32_t *d, uint16_t **size**, struct **numberType** *numDef=**buildNullNumber↩ Type**())

Construct with data array.

- **number** (const **number** &num)

  Copy constructor.
- **number** & **operator=** (const **number** &num)

  Equality constructor.
- virtual ~**number** ()

  Virtual destructor.
- void **reduce** ()

  Eliminate high-order zeros.
- void **expand** (uint16_t **size**)

  Expand number size.
- os::smart_ptr< unsigned char > **getCharData** (unsigned int &arr_len) const

  Build byte array.
- os::smart_ptr< unsigned char > **getCompCharData** (unsigned int &arr_len) const

  Build compatibility byte array.
- std::string **toString** () const

  Build hex string from number.
- void **fromString** (const std::string &str)

  Re-builds number from provided string.
- uint32_t **operator[ ]** (uint16_t pos) const

  Read-only data access.
- uint32_t & **operator[ ]** (uint16_t pos)

  Read/write data access.
- const bool **operator==** (const **number** &comp) const

  '==' comparison operator
- const bool **operator!=** (const **number** &comp) const

  '!=' comparison operator
- const bool **operator**<**=** (const **number** &comp) const

  '<=' comparison operator
- const bool **operator**>**=** (const **number** &comp) const

  '>=' comparison operator
- const bool **operator**< (const **number** &comp) const

  '<' comparison operator
- const bool **operator**> (const **number** &comp) const

  '>' comparison operator
- int **compare** (const **number** *n2) const

  Compares two numbers.
- void **addition** (const **number** *n2, **number** *result) const

  Addition function.
- void **subtraction** (const **number** *n2, **number** *result) const

  Subtraction function.
- void **rightShift** (uint16_t n2, **number** *result) const

  Right shift function.

- void **leftShift** (uint16_t n2, **number** ∗result) const

  Left shift function.
- void **multiplication** (const **number** ∗n2, **number** ∗result) const

  Multiplication function.
- void **division** (const **number** ∗n2, **number** ∗result) const

  Division function.
- void **modulo** (const **number** ∗n2, **number** ∗result) const

  Modulo function.
- void **exponentiation** (const **number** ∗n2, **number** ∗result) const

  Exponentiation function.
- void **moduloExponentiation** (const **number** ∗n2, const **number** ∗n3, **number** ∗result) const

  Modular exponentiation.
- void **gcd** (const **number** ∗n2, **number** ∗result) const

  Greatest-common-denominator function.
- void **modInverse** (const **number** ∗n2, **number** ∗result) const

  Modular-inverse function.
- **number operator**| (const **number** &op) const

  Or operator.
- **number** & **operator|=** (const **number** &op)

  Or-equals operator.
- **number operator&** (const **number** &op) const

  And operator.
- **number** & **operator&=** (const **number** &op)

  And-equals operator.
- **number operator**^ (const **number** &op) const

  X-Or operator.
- **number** & **operator**^**=** (const **number** &op)

  X-Or-equals operator.
- **number operator**~ () const

  Negate operator.
- virtual bool **checkType** () const

  Check if the number is valid.
- bool **hasCompare** () const

  Check for the 'compare' function.
- bool **hasAddition** () const

  Check for the 'addition' function.
- bool **hasSubtraction** () const

  Check for the 'subtraction' function.
- bool **hasRightShift** () const

  Check for the 'rightShift' function.
- bool **hasLeftShift** () const

  Check for the 'leftShift' function.
- bool **hasMultiplication** () const

Check for the 'multiplication' function.

- bool **hasDivision** () const

  Check for the 'division' function.
- bool **hasModulo** () const

  Check for the 'modulo' function.
- bool **hasExponentiation** () const

  Check for the 'exponentiation' function.
- bool **hasModuloExponentiation** () const

  Check for the 'moduloExponentiation' function.
- bool **hasGCD** () const

  Check for the 'gcd' function.
- bool **hasModInverse** () const

  Check for the 'modInverse' function.
- uint16_t **size** () const

  Access data size.
- uint32_t * **data** ()

  Data access.
- const uint32_t * **data** () const

  Constant data access.
- const struct **numberType** * **numberDefinition** () const

  Access number definition.
- int **typeID** () const

  Access number ID.
- std::string **name** () const

  Access number name.

Protected Member Functions

- int **_compare** (const **number** &n2) const

  Compares two numbers.

Protected Attributes

- struct **numberType** * **_numDef**

  Definition of number algorithms.
- uint16_t **_size**

  Size of the data array.
- uint32_t * **_data**

  Data array.

6.34.1 Detailed Description

Basic number definition.

This class defines the basics of all large number classes. Operators are specifically defined in each class which inherits from number.

6.34.2 Constructor & Destructor Documentation

crypto::number::number (struct **numberType** * numDef = **buildNullNumberType**())

Construct with number definition.

Parameters

in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (uint16_t size, struct **numberType** * numDef = **buildNullNumberType**())

Construct with size.

Parameters

in	*size*	Size of the number to be constructed
in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (const uint32_t * d, uint16_t size, struct **numberType** * numDef = **buildNullNumberType**())

Construct with data array.

Parameters

in	*d*	Data array to bind to this number
in	*size*	Size of the number to be constructed
in	*numDef*	Definition of number, by default **buildNullNumberType()** (p. 16)

crypto::number::number (const **number** & num)

Copy constructor.

Parameters

in	*num*	Number used to construct this

virtual crypto::number::~number ( ) [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.34.3 Member Function Documentation

int crypto::number::_compare (const **number** & n2) const [protected]

Compares two numbers.

Parameters

in	*n2*	Number to be compared against

Returns

0 if equal, 1 if greater than, -1 if less than

void crypto::number::addition (const **number** * n2, **number** * result) const

Addition function.

Preforms this+n2=result. Note that this function will only preform the addition if the number definition defines an addition function.

Parameters

in	*n2*	Number to be added
out	*result*	Result of addition

Returns

void

virtual bool crypto::number::checkType ( ) const [inline], [virtual]

Check if the number is valid.

By default, this function returns false. Numbers which inherit this class are expected to use this function to check if the number definition matches the class definition.

Returns

true if valid type, else, false

Reimplemented in **crypto::integer** (p. 130).

int crypto::number::compare (const **number** * n2) const

Compares two numbers.

Parameters

in	*n2*	Number to be compared against

Returns

0 if equal, 1 if greater than, -1 if less than

uint32_t* crypto::number::data ( ) [inline]

Data access.

Returns

crypto::number::_data (p. 194)

const uint32_t* crypto::number::data ( ) const [inline]

Constant data access.

Returns

crypto::number::_data (p. 194)

void crypto::number::division (const **number** * n2, **number** * result) const

Division function.

Preforms this/n2=result. Note that this function will only preform the division if the number definition defines an division function.

Parameters

in	*n2*	Number to be divided by
out	*result*	Result of division

Returns

void

void crypto::number::expand (uint16_t size)

Expand number size.

Parameters

in	*size*	Size of the number to be constructed

Returns

void

void crypto::number::exponentiation (const **number** * n2, **number** * result) const

Exponentiation function.

Preforms this^n2=result. Note that this function will only preform the exponentiation if the number definition defines an exponentiation function.

Parameters

in	*n2*	Number to be raised to
out	*result*	Result of exponentiation

Returns

void

void crypto::number::fromString (const std::string & str)

Re-builds number from provided string.

Parameters

in	*str*	Hex string representing number

Returns

void

void crypto::number::gcd (const **number** * n2, **number** * result) const

Greatest-common-denominator function.

Preforms GCD of this and n2=result. Note that this function will only preform the greatest-common-denominator if the number definition defines an greatest-common-denominator function.

Parameters

in	*n2*	GCD target
out	*result*	Result of greatest-common-denominator

Returns

void

os::smart_ptr<unsigned char> crypto::number::getCharData (unsigned int & arr_len) const

Build byte array.

Constructs a byte array based on the data array of this number. Useful for binary saving and packet-izing.

Parameters

out	*arr_len*	return Byte array

os::smart_ptr<unsigned char> crypto::number::getCompCharData (unsigned int & arr_len) const

Build compatibility byte array.

Constructs a byte array based on the data array of this number. First eliminates endian differences of operating systems.

Parameters

out	*arr_len*	return Byte array

bool crypto::number::hasAddition ( ) const [inline]

Check for the 'addition' function.

Returns

crypto::number::_numDef (p. 194)->addition

bool crypto::number::hasCompare ( ) const [inline]

Check for the 'compare' function.

Returns

crypto::number::_numDef (p. 194)->compare

bool crypto::number::hasDivision ( ) const [inline]

Check for the 'division' function.

Returns

crypto::number::_numDef (p. 194)->division

bool crypto::number::hasExponentiation ( ) const [inline]

Check for the 'exponentiation' function.

Returns

crypto::number::_numDef (p. 194)->exponentiation

bool crypto::number::hasGCD ( ) const [inline]

Check for the 'gcd' function.

Returns

crypto::number::_numDef (p. 194)->gcd

bool crypto::number::hasLeftShift ( ) const [inline]

Check for the 'leftShift' function.

Returns

crypto::number::_numDef (p. 194)->leftShift

bool crypto::number::hasModInverse ( ) const [inline]

Check for the 'modInverse' function.

Returns

crypto::number::_numDef (p. 194)->modInverse

bool crypto::number::hasModulo ( ) const [inline]

Check for the 'modulo' function.

Returns

crypto::number::_numDef (p. 194)->modulo

bool crypto::number::hasModuloExponentiation ( ) const [inline]

Check for the 'moduloExponentiation' function.

Returns

crypto::number::_numDef (p. 194)->moduloExponentiation

bool crypto::number::hasMultiplication ( ) const [inline]

Check for the 'multiplication' function.

Returns

crypto::number::_numDef (p. 194)->multiplication

bool crypto::number::hasRightShift ( ) const [inline]

Check for the 'rightShift' function.

Returns

crypto::number::_numDef (p. 194)->rightShift

bool crypto::number::hasSubtraction ( ) const [inline]

Check for the 'subtraction' function.

Returns

crypto::number::_numDef (p. 194)->subtraction

void crypto::number::leftShift (uint16_t n2, **number** * result) const

Left shift function.

Preforms this<<n2=result. Note that this function will only preform the shift if the number definition defines an leftShift function.

Parameters

in	*n2*	Bits to be shifted by
out	*result*	Result of shift

Returns

void

void crypto::number::modInverse (const **number** * n2, **number** * result) const

Modular-inverse function.

Preforms (this^-1)n2=result. Note that this function will only preform the modular-inverse if the number definition defines an modular-inverse function.

Parameters

in	*n2*	Number which defines the modulo space
out	*result*	Result of modular-inverse

Returns

void

void crypto::number::modulo (const **number** * n2, **number** * result) const

Modulo function.

Preforms thisn2=result. Note that this function will only preform the modulo if the number definition defines an modulo function.

Parameters

in	*n2*	Number to be moded by
out	*result*	Result of modulo

Returns

void

void crypto::number::moduloExponentiation (const **number** * n2, const **number** * n3, **number** * result) const

Modular exponentiation.

Preforms this^n2 n3=result. Note that this function will only preform the modular exponentiation if the number definition defines an modular exponentiation function.

Parameters

`in`	*n2*	Number to be raised to
`in`	*n3*	Number defines modulo space
`out`	*result*	Result of exponentiation

Returns

void

void crypto::number::multiplication (const **number** * n2, **number** * result) const

Multiplication function.

Preforms this*n2=result. Note that this function will only preform the multiplication if the number definition defines an multiplication function.

Parameters

`in`	*n2*	Number to be multiplied
`out`	*result*	Result of multiplication

Returns

void

std::string crypto::number::name ( ) const `[inline]`

Access number name.

Returns

crypto::number::_numDef (p. 194)->name

const struct **numberType*** crypto::number::numberDefinition ( ) const `[inline]`

Access number definition.

Returns

crypto::number::_numDef (p. 194)

const bool crypto::number::operator!= (const **number** & comp) const

'!=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this != comp

number crypto::number::operator& (const **number** & op) const

And operator.

Preforms bitwise and on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this & op

number& crypto::number::operator&= (const **number** & op)

And-equals operator.

Preforms bitwise and-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this = this & op

const bool crypto::number::operator< (const **number** & comp) const

'<' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this < comp

const bool crypto::number::operator<= (const **number** & comp) const

'<=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this <= comp

number& crypto::number::operator= (const **number** & num)

Equality constructor.

Parameters

in	*num*	Number used to re-build this

Returns

Reference to this

const bool crypto::number::operator== (const **number** & comp) const

'==' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this == comp

const bool crypto::number::operator> (const **number** & comp) const

'>' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this > comp

const bool crypto::number::operator>= (const **number** & comp) const

'>=' comparison operator

Parameters

in	*comp*	Number to be compared against

Returns

this >= comp

uint32_t crypto::number::operator[ ] (uint16_t pos) const

Read-only data access.

Parameters

in	*pos*	Index to access

Returns

crypto::number::_data (p. 194)[pos]

uint32_t& crypto::number::operator[ ] (uint16_t pos)

Read/write data access.

Parameters

in	*pos*	Index to access

Returns

crypto::number::_data (p. 194)[pos]

number crypto::number::operator^ (const **number** & op) const

X-Or operator.

Preforms bitwise exclusive-or on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this ^ op

number& crypto::number::operator^= (const **number** & op)

X-Or-equals operator.

Preforms bitwise exclusive-or-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this=this ^ op

number crypto::number::operator| (const **number** & op) const

Or operator.

Preforms bitwise or on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this | op

number& crypto::number::operator|= (const **number** & op)

Or-equals operator.

Preforms bitwise or-equals on the number. Note that all numbers can preform bit-wise operations on all other numbers

Parameters

in	*op*	Number preforming bitwise operation

Returns

this = this | op

number crypto::number::operator~ ( ) const

Negate operator.

Flips all bits in the number, returning a new number.

Returns

~this

void crypto::number::reduce ( )

Eliminate high-order zeros.

Returns

void

void crypto::number::rightShift (uint16_t n2, **number** * result) const

Right shift function.

Preforms this>>n2=result. Note that this function will only preform the shift if the number definition defines an rightShift function.

Parameters

in	*n2*	Bits to be shifted by
out	*result*	Result of shift

Returns

void

uint16_t crypto::number::size ( ) const [inline]

Access data size.

Returns

crypto::number::_size (p. 194)

void crypto::number::subtraction (const **number** * n2, **number** * result) const

Subtraction function.

Preforms this-n2=result. Note that this function will only preform the subtraction if the number definition defines an subtraction function.

Parameters

in	*n2*	Number to be subtracted
out	*result*	Result of subtraction

Returns

void

std::string crypto::number::toString ( ) const

Build hex string from number.

Returns

Hex string

int crypto::number::typeID ( ) const [inline]

Access number ID.

Returns

crypto::number::_numDef (p. 194)->typeID

6.34.4 Member Data Documentation

uint32_t* crypto::number::_data [protected]

Data array.

struct **numberType*** crypto::number::_numDef [protected]

Definition of number algorithms.

uint16_t crypto::number::_size [protected]

Size of the data array.

6.35 numberType Struct Reference

Number type function structure.

Public Attributes

- int **typeID**
  ID integer of the number type.
- const char * **name**
  Name of the number type.
- **compareFunction compare**
  Pointer to comparison function.
- **operatorFunction addition**
  Pointer to addition function.
- **operatorFunction subtraction**
  Pointer to subtraction function.

- **shiftFunction rightShift**

  Pointer to right-shift function.
- **shiftFunction leftShift**

  Pointer to left-shift function.
- **operatorFunction multiplication**

  Pointer to multiplication function.
- **operatorFunction division**

  Pointer to division function.
- **operatorFunction modulo**

  Pointer to modulo function.
- **operatorFunction exponentiation**

  Pointer to exponentiation function.
- **tripleCalculation moduloExponentiation**

  Pointer to modulo exponentiation function.
- **operatorFunction gcd**

  Pointer to greatest common denominator function.
- **operatorFunction modInverse**

  Pointer to modulo inverse function.

6.35.1 Detailed Description

Number type function structure.

This structure contains a series of meaningful function pointers which define functions required to meaningfully define a numerical system.

6.35.2 Member Data Documentation

operatorFunction numberType::addition

Pointer to addition function.

compareFunction numberType::compare

Pointer to comparison function.

operatorFunction numberType::division

Pointer to division function.

operatorFunction numberType::exponentiation

Pointer to exponentiation function.

operatorFunction numberType::gcd

Pointer to greatest common denominator function.

shiftFunction numberType::leftShift

Pointer to left-shift function.

operatorFunction numberType::modInverse

Pointer to modulo inverse function.

operatorFunction numberType::modulo

Pointer to modulo function.

tripleCalculation numberType::moduloExponentiation

Pointer to modulo exponentiation function.

operatorFunction numberType::multiplication

Pointer to multiplication function.

const char* numberType::name

Name of the number type.

shiftFunction numberType::rightShift

Pointer to right-shift function.

operatorFunction numberType::subtraction

Pointer to subtraction function.

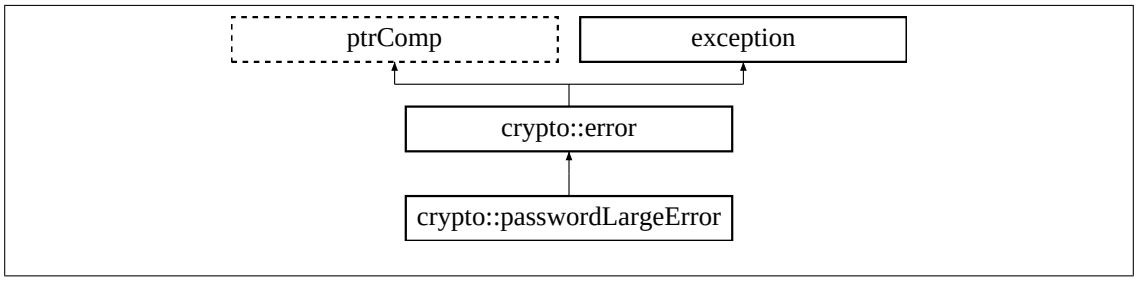
int numberType::typeID

ID integer of the number type.

6.36 crypto::passwordLargeError Class Reference

Symmetric key too big.

Inheritance diagram for crypto::passwordLargeError:



Public Member Functions

- virtual ~**passwordLargeError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Password Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Password too large".

6.36.1 Detailed Description

Symmetric key too big.

Thrown when a symmetric key is provided which is bigger than the maximum for the specific algorithm.

6.36.2 Constructor & Destructor Documentation

virtual crypto::passwordLargeError::~passwordLargeError ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.36.3 Member Function Documentation

std::string crypto::passwordLargeError::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Password too large".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::passwordLargeError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Password Size Error".

Returns

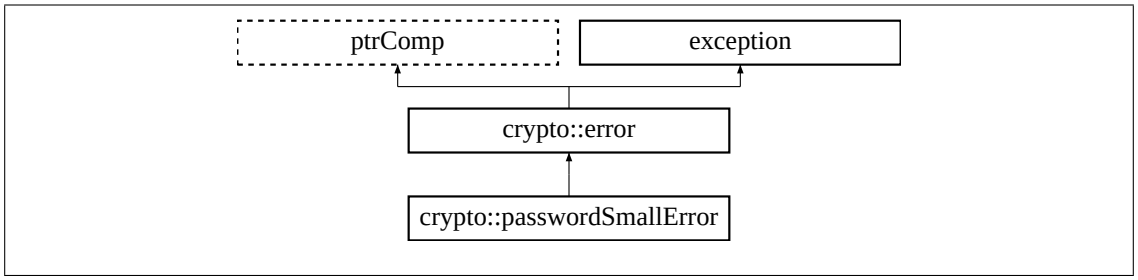
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.37 crypto::passwordSmallError Class Reference

Symmetric key too small.

Inheritance diagram for crypto::passwordSmallError:



Public Member Functions

- virtual ~**passwordSmallError** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Password Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Password too small".

6.37.1 Detailed Description

Symmetric key too small.

Thrown when a symmetric key is provided which is smaller than the minimum for the specific algorithm.

6.37.2 Constructor & Destructor Documentation

virtual crypto::passwordSmallError::~passwordSmallError ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.37.3 Member Function Documentation

std::string crypto::passwordSmallError::errorDescription ( ) const `[inline]`, `[virtual]`

Long error descriptor Returns "Password too small".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::passwordSmallError::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Password Size Error".

Returns

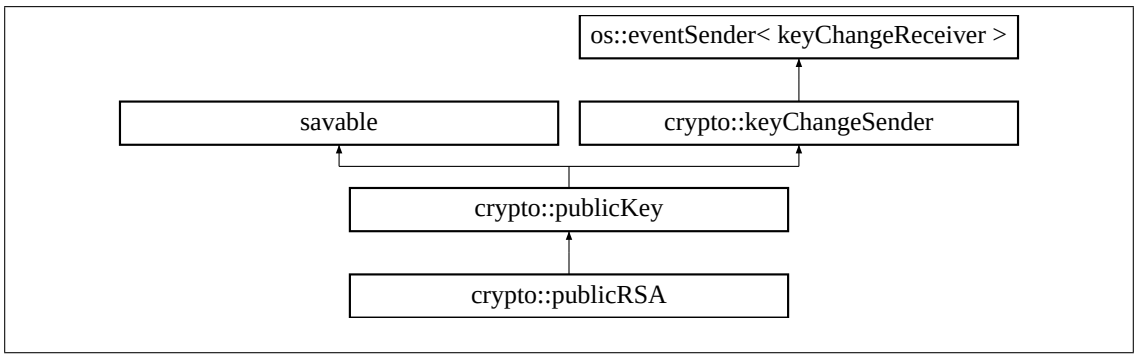
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.38 crypto::publicKey Class Reference

Base public-key class.

Inheritance diagram for crypto::publicKey:



Public Member Functions

- void **readLock** ()

  Increments the read-lock.
- void **readUnlock** ()

  Decrements the read-lock.
- virtual ~**publicKey** ()

  Virtual destructor.
- bool **searchKey** (**hash** hsh, unsigned int &hist, bool &type)

  Searches for key by hash.
- bool **searchKey** (os::smart_ptr< **number** > key, unsigned int &hist, bool &type)

  Searches for key.
- virtual os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num) const

  Converts number to correct type.
- virtual os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len) const

  Converts array to correct number type.
- virtual os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len) const

  Converts byte array to correct number type.
- os::smart_ptr< **number** > **getN** () const

Public key access.

- os::smart_ptr< **number** > **getD** () const

  Private key access.
- uint64_t **timestamp** () const

  Time-stamp access.
- os::smart_ptr< **number** > **getOldN** (unsigned int **history**=0)

  Access old public keys.
- os::smart_ptr< **number** > **getOldD** (unsigned int **history**=0)

  Access old private keys.
- uint64_t **getOldTimestamp** (unsigned int **history**=0)

  Access old time-stamps.
- virtual void **generateNewKeys** ()

  Key generation function.
- virtual bool **generating** ()

  Tests if the keys are in the process of generating.
- uint16_t **algorithm** () const

  Access algorithm ID.
- virtual std::string **algorithmName** () const

  Access algorithm name.
- uint16_t **size** () const

  Access key size.
- void **setHistory** (uint16_t hist)

  Sets history size.
- uint16_t **history** () const
- void **save** ()

  Re-save the entire structure.
- void **loadFile** ()

  Loads the structure from a file.
- void **setFileName** (std::string **fileName**)

  Set the save file name.
- void **setPassword** (unsigned char ∗key, unsigned int keyLen)

  Binds a new symmetric key.
- void **setPassword** (std::string password)
- void **setEncryptionAlgorithm** (os::smart_ptr< **streamPackageFrame** > stream_algo)

  Sets the symmetric encryption algorithm.
- const std::string & **fileName** () const

  Return the save file path.
- void **addKeyPair** (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint64_t tms=os↩ ::getTimestamp())

  Add key pair.
- virtual os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN=NULL) const

  Number encode.

- virtual void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN=NULL) const

  Data encode against number.
- virtual void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const

  Data encode.
- virtual os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code) const

  Number decode.
- virtual os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code, unsigned int hist)

  Number decode, old key.
- void **decode** (unsigned char *code, unsigned int codeLength) const

  Data decode.
- void **decode** (unsigned char *code, unsigned int codeLength, unsigned int hist)

  Data decode, old key.
- bool **operator==** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator!=** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**< (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**> (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**<**=** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.
- bool **operator**>**=** (const **publicKey** &cmp) const

  Compares equality by size and algorithm.

Static Public Member Functions

- static os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num, uint16_t **size**)

  Converts number to correct type, statically.
- static os::smart_ptr< **number** > **copyConvert** (const uint32_t *arr, uint16_t len, uint16_t **size**)

  Converts array to correct number type, statically.
- static os::smart_ptr< **number** > **copyConvert** (const unsigned char *arr, unsigned int len, uint16_t **size**)

  Converts byte array to correct number type, statically.
- static uint16_t **staticAlgorithm** ()

  Access algorithm ID.
- static std::string **staticAlgorithmName** ()

  Access algorithm name.
- static os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t **size**)

Static number encode.

- static void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Hybrid data encode against number.
- static void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength, uint16_t **size**)

  Static data encode.

Static Public Attributes

- static const unsigned int **CURRENT_INDEX** = ~0

  Current key index Allows the current key to be accessed as historical index '-1'.
- static const bool **PUBLIC** =true

  Public boolean marker.
- static const bool **PRIVATE** =false

  Private boolean marker.
- static const bool **N_MARKER** =true

  N (public) boolean marker.
- static const bool **D_MARKER** =false

  D (private) boolean marker.

Protected Member Functions

- **publicKey** (uint16_t algo, uint16_t sz=size::public512)

  No key constructor.
- **publicKey** (const **publicKey** &ky)

  Copy constructor.
- **publicKey** (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint16_t algo, uint16_t sz=size::public512, uint64_t tms=os::getTimestamp())

  Construct with keys.
- **publicKey** (uint16_t algo, std::string **fileName**, std::string password="", os::smart_ptr< **stream** **PackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- **publicKey** (uint16_t algo, std::string **fileName**, unsigned char *key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- void **writeLock** ()

  Locks the write lock.
- void **writeUnlock** ()

  Unlocks the write lock.
- int **compare** (const **publicKey** &cmp) const

  Compare this with another public key.
- void **pushOldKeys** (os::smart_ptr< **number** > **n**, os::smart_ptr< **number** > **d**, uint64_t ts)

  Bind old keys to history.

Protected Attributes

- os::smart_ptr< **number** > **n**
- os::smart_ptr< **number** > **d**
- uint64_t **_timestamp**
- os::unsortedList< **number** > **oldN**
- os::unsortedList< **number** > **oldD**
- os::unsortedList< uint64_t > **_timestamps**

Private Attributes

- uint16_t **_size**
- uint16_t **_algorithm**
- uint16_t **_history**
- unsigned char * **_key**

  Symmetric key for encryption.
- unsigned int **_keyLen**

  Length of symmetric key.
- os::smart_ptr< **streamPackageFrame** > **fePackage**
- std::string **_fileName**
- os::multiLock **keyLock**

6.38.1 Detailed Description

Base public-key class.

Class which defines the general structure of a public-private key pair. The class does not define the specifics of the algorithm.

6.38.2 Constructor & Destructor Documentation

crypto::publicKey::publicKey (uint16_t algo, uint16_t sz = `size::public512`) `[protected]`

No key constructor.

Parameters

algo	Algorithm ID
sz	Size of key, size::public512 by default

crypto::publicKey::publicKey (const **publicKey** & ky) `[protected]`

Copy constructor.

Parameters

ky	Public key to be copied

crypto::publicKey::publicKey (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint16_t algo, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`) `[protected]`

Construct with keys.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
algo	Algorithm ID
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicKey::publicKey (uint16_t algo, std::string fileName, std::string password = "", os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`) `[protected]`

Construct with path to file and password.

Parameters

algo	Algorithm ID
fileName	Name of file to find keys
password	String representing symmetric key, "" by default
stream_algo	Symmetric key encryption algorithm, NULL by default

crypto::publicKey::publicKey (uint16_t algo, std::string fileName, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`) `[protected]`

Construct with path to file and password.

Parameters

algo	Algorithm ID
fileName	Name of file to find keys
key	Symmetric key
keyLen	Length of symmetric key
stream_algo	Symmetric key encryption algorithm, NULL by default

virtual crypto::publicKey::~publicKey ( ) `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.38.3 Member Function Documentation

void crypto::publicKey::addKeyPair (os::smart_ptr< **number** > _n, os::smart_ptr< **number** > _d, uint64_t tms = `os::getTimestamp()`)

Add key pair.

Adds a key-pair and binds the current keys to the history;.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
tms	Time-stamp of the current keys, now by default

Returns

void

uint16_t crypto::publicKey::algorithm ( ) const `[inline]`

Access algorithm ID.

Returns

crypto::publicKey::_algorithm (p. 217)

virtual std::string crypto::publicKey::algorithmName ( ) const `[inline]`, `[virtual]`

Access algorithm name.

Returns

crypto::publicKey::staticAlgorithmName() (p. 217)

Reimplemented in **crypto::publicRSA** (p. 231).

int crypto::publicKey::compare (const **publicKey** & cmp) const `[protected]`

Compare this with another public key.

Compares based on the algorithm ID and size of the key. Note that this will return 0 if two public keys have the same algorithm ID and size even if they have different keys.

Parameters

`in`	*cmp*	Public key to compare against

Returns

0 if equal, 1 if greater than, -1 if less than

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const os::smart_ptr< **number** > num) const [virtual]

Converts number to correct type.

Parameters

in	*num*	Number to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 231).

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const uint32_t * arr, uint16_t len) const [virtual]

Converts array to correct number type.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 231).

virtual os::smart_ptr<**number**> crypto::publicKey::copyConvert (const unsigned char * arr, unsigned int len) const [virtual]

Converts byte array to correct number type.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented in **crypto::publicRSA** (p. 232).

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const os::smart_ptr< **number** > num, uint16_t size) [static]

Converts number to correct type, statically.

Parameters

in	*num*	Number to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const uint32_t * arr, uint16_t len, uint16_t size) [static]

Converts array to correct number type, statically.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicKey::copyConvert (const unsigned char * arr, unsigned int len, uint16_t size) [static]

Converts byte array to correct number type, statically.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

virtual os::smart_ptr<**number**> crypto::publicKey::decode (os::smart_ptr< **number** > code) const [virtual]

Number decode.

Uses the private key to decode a set of data. Re-implemented by algorithm definitions which inherit from this class.

Parameters

in	*code*	Data to be decoded

Returns

Decoded number

Reimplemented in **crypto::publicRSA** (p. 233).

virtual os::smart_ptr<**number**> crypto::publicKey::decode (os::smart_ptr< **number** > code, unsigned int hist) [virtual]

Number decode, old key.

Uses the private key to decode a set of data. Re-implemented by algorithm definitions which inherit from this class.

Parameters

in	*code*	Data to be decoded
in	*hist*	Index of historical key

Returns

Decoded number

Reimplemented in **crypto::publicRSA** (p. 233).

void crypto::publicKey::decode (unsigned char * code, unsigned int codeLength) const

Data decode.

Uses the private key to decode a set of data.

Parameters

	[in/out]	code Data to be decoded
in	*codeLength*	Length of code to be decoded

Returns

void

void crypto::publicKey::decode (unsigned char * code, unsigned int codeLength, unsigned int hist)

Data decode, old key.

Uses the private key to decode a set of data.

Parameters

	[in/out]	code Data to be decoded
in	*codeLength*	Length of code to be decoded
in	*hist*	Index of historical key

Returns

void

static os::smart_ptr<**number**> crypto::publicKey::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static number encode.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

Encoded number

static void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Hybrid data encode against number.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against, NULL by default

Returns

void

static void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength, uint16_t size) [static]

Static data encode.

This function is expected to be re-implemented for each public-key type. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*nLength*	Length of key array
in	*size*	Size of key used

Returns

void

virtual os::smart_ptr<**number**> crypto::publicKey::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN = NULL) const [virtual]

Number encode.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against, NULL by default

Returns

Encoded number

Reimplemented in **crypto::publicRSA** (p. 235).

virtual void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN = NULL) const [virtual]

Data encode against number.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented in **crypto::publicRSA** (p. 235).

virtual void crypto::publicKey::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [virtual]

Data encode.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*nLength*	Length of key array

Returns

void

Reimplemented in **crypto::publicRSA** (p. 235).

const std::string& crypto::publicKey::fileName ( ) const [inline]

Return the save file path.

Returns

crypto::publicKey::_fileName (p. 218)

virtual void crypto::publicKey::generateNewKeys ( ) [virtual]

Key generation function.

Generates new keys for the specific algorithm. This is re-implemented by every algorithm.

Returns

void

Reimplemented in **crypto::publicRSA** (p. 236).

virtual bool crypto::publicKey::generating ( ) [inline], [virtual]

Tests if the keys are in the process of generating.

Returns

True if generating new keys

Reimplemented in **crypto::publicRSA** (p. 236).

os::smart_ptr<**number**> crypto::publicKey::getD ( ) const

Private key access.

Returns

crypto::publicKey::d (p. 218)

os::smart_ptr<**number**> crypto::publicKey::getN ( ) const

Public key access.

Returns

crypto::publicKey::n (p. 219)

os::smart_ptr<**number**> crypto::publicKey::getOldD (unsigned int history = `0`)

Access old private keys.

Parameters

history	Historical index, 0 by default

Returns

Private key at given index

os::smart_ptr<**number**> crypto::publicKey::getOldN (unsigned int history = `0`)

Access old public keys.

Parameters

history	Historical index, 0 by default

Returns

Public key at given index

uint64_t crypto::publicKey::getOldTimestamp (unsigned int history = `0`)

Access old time-stamps.

Parameters

history	Historical index, 0 by default

Returns

Time-stamp at given index

uint16_t crypto::publicKey::history ( ) const [inline]

Access history size

Returns

crypto::publicKey::_history (p. 218)

void crypto::publicKey::loadFile ( )

Loads the structure from a file.

Returns

void

bool crypto::publicKey::operator!= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '!='

bool crypto::publicKey::operator< (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '<'

bool crypto::publicKey::operator<= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '<='

bool crypto::publicKey::operator== (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '=='

bool crypto::publicKey::operator> (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '>'

bool crypto::publicKey::operator>= (const **publicKey** & cmp) const [inline]

Compares equality by size and algorithm.

Returns

boolean '>='

void crypto::publicKey::pushOldKeys (os::smart_ptr< **number** > n, os::smart_ptr< **number** > d, uint64_t ts) [protected]

Bind old keys to history.

Parameters

in	*n*	Old public key
in	*d*	Old private key
in	*ts*	Old time-stamp

Returns

void

void crypto::publicKey::readLock ( ) [inline]

Increments the read-lock.

Returns

void

void crypto::publicKey::readUnlock ( ) [inline]

Decrements the read-lock.

Returns

void

void crypto::publicKey::save ( )

Re-save the entire structure.

Returns

void

bool crypto::publicKey::searchKey (**hash** hsh, unsigned int & hist, bool & type)

Searches for key by hash.
Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*hsh*	Hash of the key to be searched for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

True if the key was found, else, false

bool crypto::publicKey::searchKey (os::smart_ptr< **number** > key, unsigned int & hist, bool & type)

Searches for key.
Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*num*	Key to search for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

True if the key was found, else, false

void crypto::publicKey::setEncryptionAlgorithm (os::smart_ptr< **streamPackageFrame** > stream_algo)

Sets the symmetric encryption algorithm.

Parameters

in	*stream_algo*	Symmetric key algorithm

Returns

void

void crypto::publicKey::setFileName (std::string fileName)

Set the save file name.

Parameters

in	*fileName*	Path of save file

Returns

void

void crypto::publicKey::setHistory (uint16_t hist)

Sets history size.

Determines the number of historical keys to keep recorded. Note that keys are sorted by the order they were received into this structure, not their time-stamp.

Parameters

in	*hist*	History size to be bound

Returns

void

void crypto::publicKey::setPassword (unsigned char * key, unsigned int keyLen)

Binds a new symmetric key.

Re-binding of the symmetric key will result in a re-save event through the savable class.

Parameters

in	*key*	Symmetric key
in	*keyLen*	Length of symmetric key

Returns

void

void crypto::publicKey::setPassword (std::string password)

Binds a new symmetric key

Parameters

in	*password*	String representing the symmetric key

Returns

void

uint16_t crypto::publicKey::size ( ) const [inline]

Access key size.

Returns

crypto::publicKey::_size (p. 218)

static uint16_t crypto::publicKey::staticAlgorithm ( ) [inline], [static]

Access algorithm ID.

Returns

crypto::algo::publicNULL

static std::string crypto::publicKey::staticAlgorithmName ( ) [inline], [static]

Access algorithm name.

Returns

"NULL Public Key"

uint64_t crypto::publicKey::timestamp ( ) const [inline]

Time-stamp access.

Returns

crypto::publicKey::_timestamp (p. 218)

void crypto::publicKey::writeLock ( ) [inline], [protected]

Locks the write lock.

Returns

void

void crypto::publicKey::writeUnlock ( ) [inline], [protected]

Unlocks the write lock.

Returns

void

6.38.4 Member Data Documentation

uint16_t crypto::publicKey::_algorithm [private]

@ brief ID of algorithm used

std::string crypto::publicKey::_fileName [private]

@ brief Name of file this key is saved to

uint16_t crypto::publicKey::_history [private]

@ brief Number of historical keys to keep

unsigned char* crypto::publicKey::_key [private]

Symmetric key for encryption.

unsigned int crypto::publicKey::_keyLen [private]

Length of symmetric key.

uint16_t crypto::publicKey::_size [private]

@ brief Size of the keys used

uint64_t crypto::publicKey::_timestamp [protected]

@ brief Date/time keys created

os::unsortedList<uint64_t> crypto::publicKey::_timestamps [protected]

@ brief List of time-stamps for old pairs

const unsigned int crypto::publicKey::CURRENT_INDEX = ~0 [static]

Current key index Allows the current key to be accessed as historical index '-1'.

os::smart_ptr<**number**> crypto::publicKey::d [protected]

@ brief Private key

const bool crypto::publicKey::D_MARKER =false [static]

D (private) boolean marker.

os::smart_ptr<**streamPackageFrame**> crypto::publicKey::fePackage [private]

@ brief Algorithm used for encryption

os::multiLock crypto::publicKey::keyLock [private]

@ brief Mutex for replacing the keys

os::smart_ptr<**number**> crypto::publicKey::n [protected]

@ brief Public key

const bool crypto::publicKey::N_MARKER =true [static]

N (public) boolean marker.

os::unsortedList<**number**> crypto::publicKey::oldD [protected]

@ brief List of old private keys

os::unsortedList<**number**> crypto::publicKey::oldN [protected]

@ brief List of old public keys

const bool crypto::publicKey::PRIVATE =false [static]

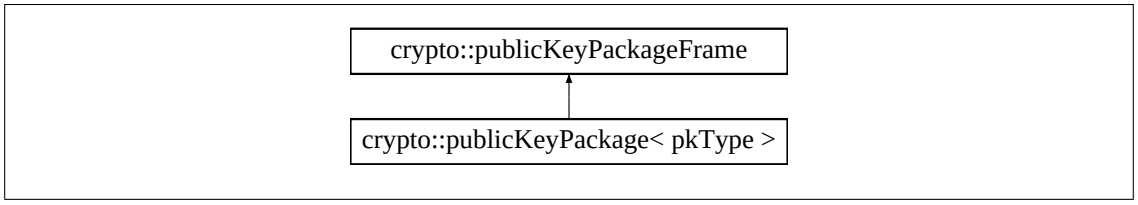
Private boolean marker.

const bool crypto::publicKey::PUBLIC =true [static]

Public boolean marker.

6.39 crypto::publicKeyPackage< pkType > Class Template Reference

Inheritance diagram for crypto::publicKeyPackage< pkType >:



Public Member Functions

- **publicKeyPackage** (uint16_t publicSize=size::public512)
- virtual ~**publicKeyPackage** ()
- os::smart_ptr< **publicKeyPackageFrame** > **getCopy** () const
- os::smart_ptr< **number** > **convert** (uint32_t *arr, uint16_t len) const
- os::smart_ptr< **number** > **convert** (const unsigned char *arr, unsigned int len) const
- os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const
- void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const
- void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const

- os::smart_ptr< **publicKey** > **generate** () const
- os::smart_ptr< **publicKey** > **bindKeys** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const
- os::smart_ptr< **publicKey** > **bindKeys** (uint32_t *_n, uint32_t *_d) const
- os::smart_ptr< **publicKey** > **openFile** (std::string fileName, std::string password) const
- os::smart_ptr< **publicKey** > **openFile** (std::string fileName, unsigned char *key, unsigned int keyLen) const
- std::string **algorithmName** () const
- uint16_t **algorithm** () const

Additional Inherited Members

6.39.1 Constructor & Destructor Documentation

template<class pkType > **crypto::publicKeyPackage**< pkType >::**publicKeyPackage** (uint16_t publicSize = `size::public512`) `[inline]`

template<class pkType > virtual **crypto::publicKeyPackage**< pkType >::~**publicKeyPackage** ( ) `[inline]`, `[virtual]`

6.39.2 Member Function Documentation

template<class pkType > uint16_t **crypto::publicKeyPackage**< pkType >::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 222).

template<class pkType > std::string **crypto::publicKeyPackage**< pkType >::algorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::bindKeys (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::bindKeys (uint32_t * _n, uint32_t * _d) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::convert (uint32_t * arr, uint16_t len) const `[inline]`, `[virtual]`

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::convert (const unsigned char * arr, unsigned int len) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**number**> **crypto::publicKeyPackage**< pkType >::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > void **crypto::publicKeyPackage**< pkType >::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > void **crypto::publicKeyPackage**< pkType >::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::generate ( ) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**publicKeyPackageFrame**> **crypto::publicKeyPackage**< pkType >::getCopy ( ) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 223).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::openFile (std::string fileName, std::string password) const [inline], [virtual]

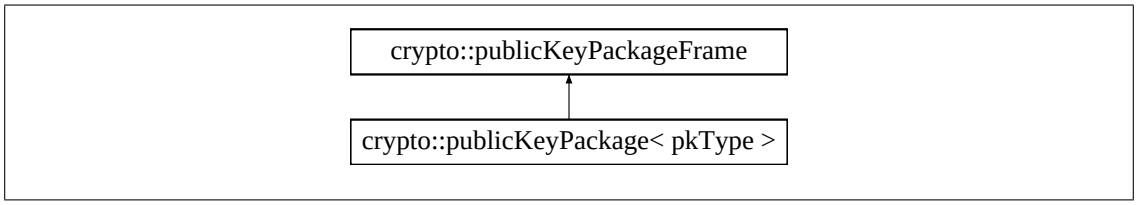
Reimplemented from **crypto::publicKeyPackageFrame** (p. 224).

template<class pkType > os::smart_ptr<**publicKey**> **crypto::publicKeyPackage**< pkType >::openFile (std::string fileName, unsigned char * key, unsigned int keyLen) const [inline], [virtual]

Reimplemented from **crypto::publicKeyPackageFrame** (p. 224).

6.40 crypto::publicKeyPackageFrame Class Reference

Inheritance diagram for crypto::publicKeyPackageFrame:



Public Member Functions

- **publicKeyPackageFrame** (uint16_t publicSize=size::public512)
- virtual ~**publicKeyPackageFrame** ()
- virtual os::smart_ptr< **publicKeyPackageFrame** > **getCopy** () const
- virtual os::smart_ptr< **number** > **convert** (uint32_t *arr, uint16_t len) const
- virtual os::smart_ptr< **number** > **convert** (const unsigned char *arr, unsigned int len) const
- virtual os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const
- virtual void **encode** (unsigned char *code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const
- virtual void **encode** (unsigned char *code, unsigned int codeLength, unsigned const char *publicN, unsigned int nLength) const
- virtual os::smart_ptr< **publicKey** > **generate** () const
- virtual os::smart_ptr< **publicKey** > **bindKeys** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const
- virtual os::smart_ptr< **publicKey** > **bindKeys** (uint32_t *_n, uint32_t *_d) const
- virtual os::smart_ptr< **publicKey** > **openFile** (std::string fileName, std::string password) const
- virtual os::smart_ptr< **publicKey** > **openFile** (std::string fileName, unsigned char *key, unsigned int keyLen) const
- virtual std::string **algorithmName** () const
- virtual uint16_t **algorithm** () const
- void **setKeySize** (uint16_t publicSize)
- uint16_t **keySize** () const

Protected Attributes

- uint16_t **_publicSize**

6.40.1 Constructor & Destructor Documentation

crypto::publicKeyPackageFrame::publicKeyPackageFrame (uint16_t publicSize = `size::public512`) `[inline]`

virtual crypto::publicKeyPackageFrame::~publicKeyPackageFrame ( ) `[inline]`, `[virtual]`

6.40.2 Member Function Documentation

virtual uint16_t crypto::publicKeyPackageFrame::algorithm ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 220).

virtual std::string crypto::publicKeyPackageFrame::algorithmName ( ) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 220).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::bindKeys (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 220).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::bindKeys (uint32_t * _n, uint32_t * _d) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 220).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::convert (uint32_t * arr, uint16_t len) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 220).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::convert (const unsigned char * arr, unsigned int len) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual os::smart_ptr<**number**> crypto::publicKeyPackageFrame::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual void crypto::publicKeyPackageFrame::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual void crypto::publicKeyPackageFrame::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::generate ( ) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyPackageFrame::getCopy ( ) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

uint16_t crypto::publicKeyPackageFrame::keySize ( ) const [inline]

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::openFile (std::string fileName, std::string password) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

virtual os::smart_ptr<**publicKey**> crypto::publicKeyPackageFrame::openFile (std::string fileName, unsigned char * key, unsigned int keyLen) const [inline], [virtual]

Reimplemented in **crypto::publicKeyPackage**< **pkType** > (p. 221).

void crypto::publicKeyPackageFrame::setKeySize (uint16_t publicSize) [inline]

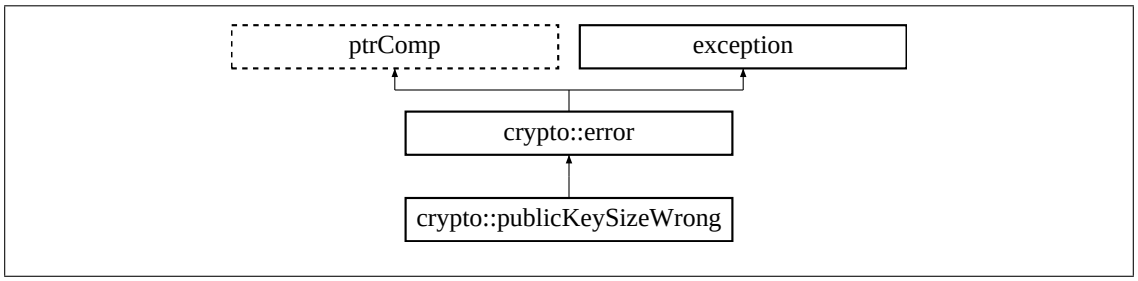
6.40.3 Member Data Documentation

uint16_t crypto::publicKeyPackageFrame::_publicSize [protected]

6.41 crypto::publicKeySizeWrong Class Reference

Public-key size error.

Inheritance diagram for crypto::publicKeySizeWrong:



Public Member Functions

- virtual ~**publicKeySizeWrong** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Public Key Size Wrong".
- std::string **errorDescription** () const

  Long error descriptor Returns "Attempted to use a code or n of improper size".

6.41.1 Detailed Description

Public-key size error.

Thrown when a public key or public key interaction detects a size mis-match or illegal size.

6.41.2 Constructor & Destructor Documentation

virtual crypto::publicKeySizeWrong::~publicKeySizeWrong ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.41.3 Member Function Documentation

std::string crypto::publicKeySizeWrong::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Attempted to use a code or n of improper size".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::publicKeySizeWrong::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Public Key Size Wrong".

Returns

Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.42 crypto::publicKeyTypeBank Class Reference

Public Member Functions

- virtual ~**publicKeyTypeBank** ()
- void **setDefaultPackage** (os::smart_ptr< **publicKeyPackageFrame** > package)
- const os::smart_ptr< **publicKeyPackageFrame** > **defaultPackage** () const
- void **pushPackage** (os::smart_ptr< **publicKeyPackageFrame** > package)
- const os::smart_ptr< **publicKeyPackageFrame** > **findPublicKey** (uint16_t pkID) const
- const os::smart_ptr< **publicKeyPackageFrame** > **findPublicKey** (const std::string &pkName) const

Static Public Member Functions

- static os::smart_ptr< **publicKeyTypeBank** > **singleton** ()

Private Member Functions

- **publicKeyTypeBank** ()

Private Attributes

- os::smart_ptr< **publicKeyPackageFrame** > **_defaultPackage**
- std::vector< os::smart_ptr< **publicKeyPackageFrame** > > **packageVector**

6.42.1 Constructor & Destructor Documentation

crypto::publicKeyTypeBank::publicKeyTypeBank ( ) [private]

virtual crypto::publicKeyTypeBank::~publicKeyTypeBank ( ) [inline], [virtual]

6.42.2 Member Function Documentation

const os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::defaultPackage ( ) const [inline]

const os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::findPublicKey (uint16_t pkID) const

const os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::findPublicKey (const std::string & pkName) const

void crypto::publicKeyTypeBank::pushPackage (os::smart_ptr< **publicKeyPackageFrame** > package)

void crypto::publicKeyTypeBank::setDefaultPackage (os::smart_ptr< **publicKeyPackageFrame** > package)

static os::smart_ptr<**publicKeyTypeBank**> crypto::publicKeyTypeBank::singleton ( ) [static]

6.42.3 Member Data Documentation

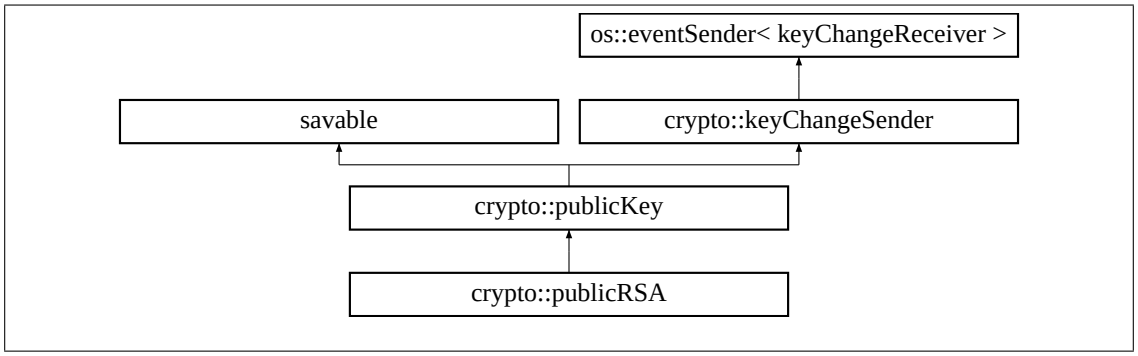
os::smart_ptr<**publicKeyPackageFrame**> crypto::publicKeyTypeBank::_defaultPackage [private]

std::vector<os::smart_ptr<**publicKeyPackageFrame**> > crypto::publicKeyTypeBank::packageVector [private]

6.43 crypto::publicRSA Class Reference

RSA public-key encryption.

Inheritance diagram for crypto::publicRSA:



Public Member Functions

- **publicRSA** (uint16_t sz=size::public256)

  Default RSA constructor.
- **publicRSA** (**publicRSA** &ky)

  Copy Constructor.
- **publicRSA** (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d, uint16_t sz=size↩ ::public512, uint64_t tms=os::getTimestamp())

  Construct with keys.
- **publicRSA** (uint32_t ∗_n, uint32_t ∗_d, uint16_t sz=size::public512, uint64_t tms=os::get↩ Timestamp())

  Construct with key arrays.
- **publicRSA** (std::string **fileName**, std::string password="", os::smart_ptr< **streamPackage**↩ **Frame** > stream_algo=NULL)

  Construct with path to file and password.
- **publicRSA** (std::string **fileName**, unsigned char ∗key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo=NULL)

  Construct with path to file and password.
- virtual ~**publicRSA** ()

  Virtual destructor.
- os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num) const

  Converts number to integer.
- os::smart_ptr< **number** > **copyConvert** (const uint32_t ∗arr, uint16_t len) const

  Converts array to integer.
- os::smart_ptr< **number** > **copyConvert** (const unsigned char ∗arr, unsigned int len) const

  Converts byte array to integer.
- std::string **algorithmName** () const

  Access algorithm name.
- bool **generating** ()

  Tests if the keys are in the process of generating.
- void **generateNewKeys** ()

  Key generation function.

- os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN=NULL) const

  Number encode.
- void **encode** (unsigned char ∗code, unsigned int codeLength, os::smart_ptr< **number** > publicN=N↩ ULL) const

  Hybrid data encode against number.
- void **encode** (unsigned char ∗code, unsigned int codeLength, unsigned const char ∗publicN, unsigned int nLength) const

  Data encode against number.
- os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code) const

  Number decode.
- os::smart_ptr< **number** > **decode** (os::smart_ptr< **number** > code, unsigned int hist)

  Old number decode.

Static Public Member Functions

- static os::smart_ptr< **number** > **copyConvert** (const os::smart_ptr< **number** > num, uint16_t **size**)

  Converts number to integer, statically.
- static os::smart_ptr< **number** > **copyConvert** (const uint32_t ∗arr, uint16_t len, uint16_t **size**)

  Converts array to integer, statically.
- static os::smart_ptr< **number** > **copyConvert** (const unsigned char ∗arr, unsigned int len, uint16_t **size**)

  Converts byte array to integer, statically.
- static uint16_t **staticAlgorithm** ()

  Access algorithm ID.
- static std::string **staticAlgorithmName** ()

  Access algorithm name.
- static os::smart_ptr< **number** > **encode** (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Static number encode.
- static void **encode** (unsigned char ∗code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t **size**)

  Static data encode.
- static void **encode** (unsigned char ∗code, unsigned int codeLength, unsigned const char ∗publicN, unsigned int nLength, uint16_t **size**)

  Static data encode.

Private Member Functions

- void **initE** ()

  *Subroutine initializing **crypto::publicRSA::e*** (p. *237*).

Private Attributes

- **integer e**

  Used in intermediate calculation.
- os::smart_ptr< **RSAKeyGenerator** > **keyGen**

  Key generation class.

Friends

- class **RSAKeyGenerator**

  Friendship with key generation.

Additional Inherited Members

6.43.1 Detailed Description

RSA public-key encryption.
This class defines an RSA algorithm for public-key cryptography.

6.43.2 Constructor & Destructor Documentation

crypto::publicRSA::publicRSA (uint16_t sz = `size::public256`)

Default RSA constructor.
Initializes and generates keys for a new pair of RSA keys. This serves as the default constructor for RSA keys.

Parameters

`in`	*sz*	Size of keys, crypto::size::public256 by default

crypto::publicRSA::publicRSA (**publicRSA** & ky)

Copy Constructor.
Copies the keys in one RSA pair into another. This copying includes all historical records as well.

Parameters

`in`	*ky*	Key pair to be copied

crypto::publicRSA::publicRSA (os::smart_ptr< **integer** > _n, os::smart_ptr< **integer** > _d, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`)

Construct with keys.

Parameters

_n	Smart pointer to public key
_d	Smart pointer to private key
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicRSA::publicRSA (uint32_t * _n, uint32_t * _d, uint16_t sz = `size::public512`, uint64_t tms = `os::getTimestamp()`)

Construct with key arrays.

Parameters

_n	Array of public key
_d	Array of private key
sz	Size of key, size::public512 by default
tms	Time-stamp of the current keys, now by default

crypto::publicRSA::publicRSA (std::string fileName, std::string password = "", os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with path to file and password.

Parameters

fileName	Name of file to find keys
password	String representing symmetric key, "" by default
stream_algo	Symmetric key encryption algorithm, NULL by default

crypto::publicRSA::publicRSA (std::string fileName, unsigned char * key, unsigned int keyLen, os::smart_ptr< **streamPackageFrame** > stream_algo = `NULL`)

Construct with path to file and password.

Parameters

fileName	Name of file to find keys
key	Symmetric key
keyLen	Length of symmetric key
stream_algo	Symmetric key encryption algorithm, NULL by default

virtual crypto::publicRSA::~publicRSA ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.43.3 Member Function Documentation

std::string crypto::publicRSA::algorithmName ( ) const [inline], [virtual]

Access algorithm name.

Returns

crypto::publicRSA::staticAlgorithmName() (p. 236)

Reimplemented from **crypto::publicKey** (p. 205).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const os::smart_ptr< **number** > num) const [virtual]

Converts number to integer.

Parameters

in	*num*	Number to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 206).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const uint32_t * arr, uint16_t len) const [virtual]

Converts array to integer.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 206).

os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const unsigned char * arr, unsigned int len) const [virtual]

Converts byte array to integer.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

Reimplemented from **crypto::publicKey** (p. 206).

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const os::smart_ptr< **number** > num, uint16_t size) [static]

Converts number to integer, statically.

Parameters

in	*num*	Number to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const uint32_t * arr, uint16_t len, uint16_t size) [static]

Converts array to integer, statically.

Parameters

in	*arr*	Array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

static os::smart_ptr<**number**> crypto::publicRSA::copyConvert (const unsigned char * arr, unsigned int len, uint16_t size) [static]

Converts byte array to integer, statically.

Parameters

in	*arr*	Byte array to be converted
in	*len*	Length of array to be converted

Returns

Converted number

os::smart_ptr<**number**> crypto::publicRSA::decode (os::smart_ptr< **number** > code) const [virtual]

Number decode.

Uses the private key to decode a set of data based on the RSA algorithm.

Parameters

in	*code*	Data to be decoded

Returns

Decoded number

Reimplemented from **crypto::publicKey** (p. 207).

os::smart_ptr<**number**> crypto::publicRSA::decode (os::smart_ptr< **number** > code, unsigned int hist) [virtual]

Old number decode.

Uses old private keys to decode a set of data based on the RSA algorithm.

Parameters

in	*code*	Data to be decoded
in	*hist*	Index of historical key

Returns

Decoded number

Reimplemented from **crypto::publicKey** (p. 208).

static os::smart_ptr<**number**> crypto::publicRSA::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static number encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

in	*code*	Data to be encoded
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

Encoded number

static void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN, uint16_t size) [static]

Static data encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*size*	Size of key used

Returns

void

static void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength, uint16_t size) [static]

Static data encode.

Encodes based on the RSA algorithm. This function must be static because data can be encoded with a public key even though a node does not have its own keys defined.

Parameters

	[in/out]	code Data to be encoded
in	*codeLength*	Length of code array
in	*publicN*	Public key to be encoded against
in	*nLength*	Length of key array
in	*size*	Size of key used

Returns

void

os::smart_ptr<**number**> crypto::publicRSA::encode (os::smart_ptr< **number** > code, os::smart_ptr< **number** > publicN = `NULL`) const `[virtual]`

Number encode.

Parameters

`in`	*code*	Data to be encoded
`in`	*publicN*	Public key to be encoded against, NULL by default

Returns

Encoded number

Reimplemented from **crypto::publicKey** (p. 210).

void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, os::smart_ptr< **number** > publicN = `NULL`) const `[virtual]`

Hybrid data encode against number.

Parameters

	[in/out]	code Data to be encoded
`in`	*codeLength*	Length of code array
`in`	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented from **crypto::publicKey** (p. 210).

void crypto::publicRSA::encode (unsigned char * code, unsigned int codeLength, unsigned const char * publicN, unsigned int nLength) const `[virtual]`

Data encode against number.

Parameters

	[in/out]	code Data to be encoded
`in`	*codeLength*	Length of code array
`in`	*publicN*	Public key to be encoded against, NULL by default

Returns

void

Reimplemented from **crypto::publicKey** (p. 211).

void crypto::publicRSA::generateNewKeys ( ) [virtual]

Key generation function.

Generates new keys for the specific algorithm. This is re-implemented by every algorithm.

Returns

void

Reimplemented from **crypto::publicKey** (p. 211).

bool crypto::publicRSA::generating ( ) [virtual]

Tests if the keys are in the process of generating.

Returns

True if generating new keys

Reimplemented from **crypto::publicKey** (p. 211).

void crypto::publicRSA::initE ( ) [private]

Subroutine initializing **crypto::publicRSA::e** (p. 237).

static uint16_t crypto::publicRSA::staticAlgorithm ( ) [inline], [static]

Access algorithm ID.

Returns

crypto::algo::publicRSA

static std::string crypto::publicRSA::staticAlgorithmName ( ) [inline], [static]

Access algorithm name.

Returns

"RSA"

6.43.4 Friends And Related Function Documentation

friend class **RSAKeyGenerator** [friend]

Friendship with key generation.

The **crypto::RSAKeyGenerator** (p. 243) must be able to access the private members of the RSA public key class to bind newly generated keys.

6.43.5 Member Data Documentation

integer crypto::publicRSA::e `[private]`

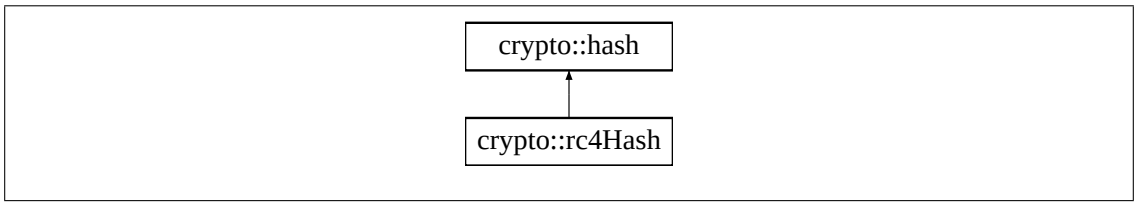
Used in intermediate calculation.

os::smart_ptr<**RSAKeyGenerator**> crypto::publicRSA::keyGen `[private]`

Key generation class.
This pointer will be NULL unless a key is currently being generated/

6.44 crypto::rc4Hash Class Reference

RC-4 hash class.
Inheritance diagram for crypto::rc4Hash:



Public Member Functions

- **rc4Hash** ()
  Default RC-4 hash constructor.
- **rc4Hash** (const unsigned char ∗**data**, uint16_t **size**)
  Raw data copy.
- **rc4Hash** (const **rc4Hash** &cpy)
  RC-4 copy constructor.
- void **preformHash** (const unsigned char ∗**data**, uint32_t dLen)
  Binds a data-set.
- std::string **algorithmName** () const
  Algorithm name string access.

Static Public Member Functions

- static std::string **staticAlgorithmName** ()
  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()
  Algorithm ID number access.
- static **rc4Hash hash64Bit** (const unsigned char ∗**data**, uint32_t length)
  Static 64 bit hash.
- static **rc4Hash hash128Bit** (const unsigned char ∗**data**, uint32_t length)

Static 128 bit hash.

- static **rc4Hash hash256Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 256 bit hash.

- static **rc4Hash hash512Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 512 bit hash.

Private Member Functions

- **rc4Hash** (const unsigned char ∗**data**, uint32_t length, uint16_t **size**)

  RC-4 hash constructor.

Additional Inherited Members

6.44.1 Detailed Description

RC-4 hash class.

This class defines an RC-4 based hash. Note that this hash is likely cryptographically secure, but not proven cryptographically secure.

6.44.2 Constructor & Destructor Documentation

crypto::rc4Hash::rc4Hash (const unsigned char ∗ data, uint32_t length, uint16_t size) `[private]`

RC-4 hash constructor.

Constructs a hash with the data to be hashed, the length of the array and the size of the hash to be constructed.

Parameters

`in`	*data*	Data array
`in`	*length*	Length of data array
`in`	*size*	Size of hash

crypto::rc4Hash::rc4Hash ( ) `[inline]`

Default RC-4 hash constructor.

Constructs an empty RC-4 hash class.

crypto::rc4Hash::rc4Hash (const unsigned char ∗ data, uint16_t size)

Raw data copy.

Initializes the RC-4 hash with a data array. This data array is not hashed but assumed to represent hashed data.

Parameters

in	*data*	Hashed data array
in	*size*	Size of hash array

crypto::rc4Hash::rc4Hash (const **rc4Hash** & cpy) [inline]

RC-4 copy constructor.

Constructs an RC-4 hash with another RC-4 hash.

Parameters

in	*cpy*	Hash to be copied

6.44.3 Member Function Documentation

std::string crypto::rc4Hash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.

Returns the name of the current algorithm string. This function requires an instantiated RC-4 hash.

Returns

"RC-4"

Reimplemented from **crypto::hash** (p. 118).

static **rc4Hash** crypto::rc4Hash::hash128Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 128 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 128 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **rc4Hash** crypto::rc4Hash::hash256Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 256 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 256 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **rc4Hash** crypto::rc4Hash::hash512Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 512 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 512 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **rc4Hash** crypto::rc4Hash::hash64Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 64 bit hash.

Hashes the provided data array with the RC-4 algorithm, returning a 64 bit RC-4 hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

void crypto::rc4Hash::preformHash (const unsigned char * data, uint32_t dLen)

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

static uint16_t crypto::rc4Hash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.

Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashRC4

static std::string crypto::rc4Hash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.

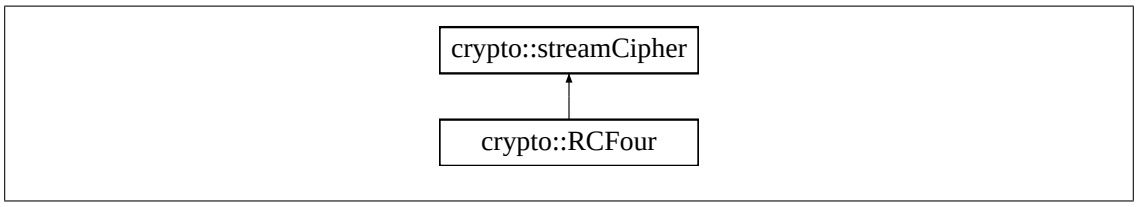
Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"RC-4"

6.45 crypto::RCFour Class Reference

Inheritance diagram for crypto::RCFour:



Public Member Functions

- **RCFour** (uint8_t *arr, int len)
- virtual ~**RCFour** ()
- uint8_t **getNext** ()
- uint16_t **algorithm** () const
- const std::string **algorithmName** () const

Static Public Member Functions

- static uint16_t **staticAlgorithm** ()
- static std::string **staticAlgorithmName** ()

Private Attributes

- uint8_t * **SArray**
- int **i**
- int **j**
- int **u**

6.45.1 Constructor & Destructor Documentation

crypto::RCFour::RCFour (uint8_t * arr, int len)

virtual crypto::RCFour::~RCFour ( ) [virtual]

6.45.2 Member Function Documentation

uint16_t crypto::RCFour::algorithm ( ) const [inline], [virtual]

Reimplemented from **crypto::streamCipher** (p. 245).

const std::string crypto::RCFour::algorithmName ( ) const [inline], [virtual]

Reimplemented from **crypto::streamCipher** (p. 246).

uint8_t crypto::RCFour::getNext ( ) [virtual]

Reimplemented from **crypto::streamCipher** (p. 246).

static uint16_t crypto::RCFour::staticAlgorithm ( ) [inline], [static]

static std::string crypto::RCFour::staticAlgorithmName ( ) [inline], [static]

6.45.3 Member Data Documentation

int crypto::RCFour::i [private]

int crypto::RCFour::j [private]

uint8_t* crypto::RCFour::SArray [private]

int crypto::RCFour::u [private]

6.46 crypto::RSAKeyGenerator Class Reference

Helper key generation class.

Public Member Functions

- **RSAKeyGenerator** (**publicRSA** &m)
  Constructs a generator with an RSA key.
- virtual ~**RSAKeyGenerator** ()
  Virtual destructor.
- **integer generatePrime** ()
  Generates a prime number.
- void **pushValues** ()
  Bind generated keys to master.

Public Attributes

- **integer p**

  Intermediate prime.
- **integer q**

  Intermediate prime.

Private Attributes

- **publicRSA** ∗ **master**

  Pointer to keys.

6.46.1 Detailed Description

Helper key generation class.

This class helps to generate RSA keys. Once keys are generated, this class is destroyed.

6.46.2 Constructor & Destructor Documentation

crypto::RSAKeyGenerator::RSAKeyGenerator (**publicRSA** & m)

Constructs a generator with an RSA key.

This class is meaningless without a a reference to an RSA key to bind newly created keys to.

virtual crypto::RSAKeyGenerator::~RSAKeyGenerator ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.46.3 Member Function Documentation

integer crypto::RSAKeyGenerator::generatePrime ( )

Generates a prime number.

Returns

Prime integer

void crypto::RSAKeyGenerator::pushValues ( )

Bind generated keys to master.

Returns

void

6.46.4 Member Data Documentation

publicRSA∗ crypto::RSAKeyGenerator::master [private]

Pointer to keys.

Points to the RSA keys this generator will be placing its generated keys into.

integer crypto::RSAKeyGenerator::p

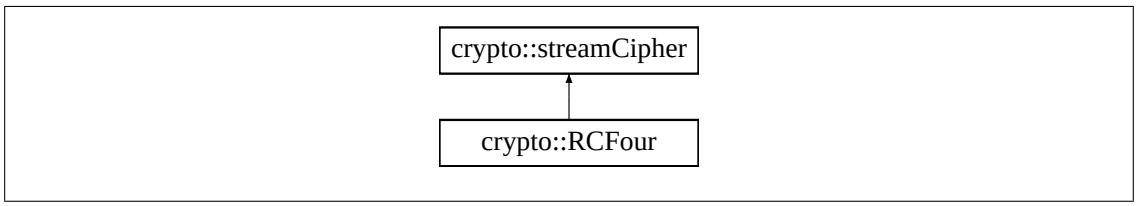
Intermediate prime.

integer crypto::RSAKeyGenerator::q

Intermediate prime.

6.47 crypto::streamCipher Class Reference

Inheritance diagram for crypto::streamCipher:



Public Member Functions

- virtual ~**streamCipher** ()
- virtual uint8_t **getNext** ()
- virtual uint16_t **algorithm** () const
- virtual const std::string **algorithmName** () const

Static Public Member Functions

- static uint16_t **staticAlgorithm** ()
- static std::string **staticAlgorithmName** ()

6.47.1 Constructor & Destructor Documentation

virtual crypto::streamCipher::~streamCipher ( ) [inline], [virtual]

6.47.2 Member Function Documentation

virtual uint16_t crypto::streamCipher::algorithm ( ) const [inline], [virtual]

Reimplemented in **crypto::RCFour** (p. 243).

virtual const std::string crypto::streamCipher::algorithmName ( ) const [inline], [virtual]

Reimplemented in **crypto::RCFour** (p. 243).

virtual uint8_t crypto::streamCipher::getNext ( ) [inline], [virtual]

Reimplemented in **crypto::RCFour** (p. 243).

static uint16_t crypto::streamCipher::staticAlgorithm ( ) [inline], [static]

static std::string crypto::streamCipher::staticAlgorithmName ( ) [inline], [static]

6.48 crypto::streamDecrypter Class Reference

Public Member Functions

- **streamDecrypter** (os::smart_ptr< **streamCipher** > c)
- virtual ~**streamDecrypter** ()
- uint8_t * **recieveData** (uint8_t *array, unsigned int len, uint16_t flag)

Private Attributes

- os::smart_ptr< **streamCipher** > **cipher**
- **streamPacket** ** **packetArray**
- unsigned int **last_value**
- unsigned int **mid_value**

6.48.1 Constructor & Destructor Documentation

crypto::streamDecrypter::streamDecrypter (os::smart_ptr< **streamCipher** > c)

virtual crypto::streamDecrypter::~streamDecrypter ( ) [virtual]

6.48.2 Member Function Documentation

uint8_t* crypto::streamDecrypter::recieveData (uint8_t * array, unsigned int len, uint16_t flag)

6.48.3 Member Data Documentation

os::smart_ptr<**streamCipher**> crypto::streamDecrypter::cipher [private]

unsigned int crypto::streamDecrypter::last_value [private]

unsigned int crypto::streamDecrypter::mid_value [private]

streamPacket** crypto::streamDecrypter::packetArray [private]

6.49 crypto::streamEncrypter Class Reference

Public Member Functions

- **streamEncrypter** (os::smart_ptr< **streamCipher** > c)

- virtual ~**streamEncrypter** ()
- uint8_t * **sendData** (uint8_t *array, unsigned int len, uint16_t &flag)

Private Attributes

- os::smart_ptr< **streamCipher** > **cipher**
- unsigned int **last_loc**
- uint16_t * **ID_check**

6.49.1 Constructor & Destructor Documentation

crypto::streamEncrypter::streamEncrypter (os::smart_ptr< **streamCipher** > c)

virtual crypto::streamEncrypter::~streamEncrypter ( ) [virtual]

6.49.2 Member Function Documentation

uint8_t* crypto::streamEncrypter::sendData (uint8_t * array, unsigned int len, uint16_t & flag)

6.49.3 Member Data Documentation

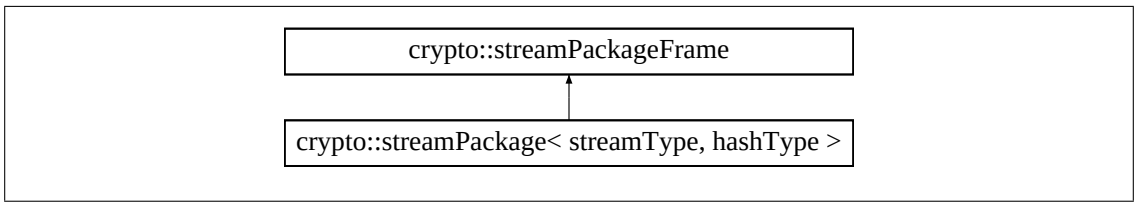
os::smart_ptr<**streamCipher**> crypto::streamEncrypter::cipher [private]

uint16_t* crypto::streamEncrypter::ID_check [private]

unsigned int crypto::streamEncrypter::last_loc [private]

6.50 crypto::streamPackage< streamType, hashType > Class Template Reference

Inheritance diagram for crypto::streamPackage< streamType, hashType >:



Public Member Functions

- **streamPackage** (uint16_t **hashSize**=size::hash256)
- virtual ~**streamPackage** ()
- os::smart_ptr< **streamPackageFrame** > **getCopy** () const
- **hash hashEmpty** () const
- **hash hashData** (unsigned char *data, uint32_t len) const
- **hash hashCopy** (unsigned char *data) const
- os::smart_ptr< **streamCipher** > **buildStream** (unsigned char *data, uint32_t len) const
- std::string **streamAlgorithmName** () const

- uint16_t **streamAlgorithm** () const
- std::string **hashAlgorithmName** () const
- uint16_t **hashAlgorithm** () const

Additional Inherited Members

6.50.1 Constructor & Destructor Documentation

template<class streamType , class hashType > **crypto::streamPackage**< streamType, hashType >::**streamPackage** (uint16_t hashSize = `size::hash256`) `[inline]`

template<class streamType , class hashType > virtual **crypto::streamPackage**< streamType, hashType >::~**streamPackage** ( ) `[inline]`, `[virtual]`

6.50.2 Member Function Documentation

template<class streamType , class hashType > os::smart_ptr<**streamCipher**> **crypto::streamPackage**< streamType, hashType >::buildStream (unsigned char * data, uint32_t len) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > os::smart_ptr<**streamPackageFrame**> **crypto::streamPackage**< streamType, hashType >::getCopy ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > uint16_t **crypto::streamPackage**< streamType, hashType >::hashAlgorithm ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > std::string **crypto::streamPackage**< streamType, hashType >::hashAlgorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashCopy (unsigned char * data) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashData (unsigned char * data, uint32_t len) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > **hash crypto::streamPackage**< streamType, hashType >::hashEmpty ( ) const `[inline]`, `[virtual]`

Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > uint16_t **crypto::streamPackage**< streamType, hashType >::streamAlgorithm ( ) const [inline], [virtual]

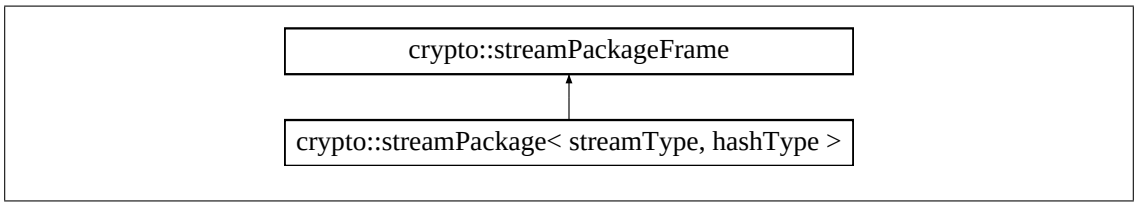
Reimplemented from **crypto::streamPackageFrame** (p. 250).

template<class streamType , class hashType > std::string **crypto::streamPackage**< streamType, hashType >::streamAlgorithmName ( ) const [inline], [virtual]

Reimplemented from **crypto::streamPackageFrame** (p. 251).

6.51 crypto::streamPackageFrame Class Reference

Inheritance diagram for crypto::streamPackageFrame:



Public Member Functions

- **streamPackageFrame** (uint16_t **hashSize**=size::hash256)
- virtual ~**streamPackageFrame** ()
- virtual os::smart_ptr< **streamPackageFrame** > **getCopy** () const
- virtual **hash hashEmpty** () const
- virtual **hash hashData** (unsigned char *data, uint32_t len) const
- virtual **hash hashCopy** (unsigned char *data) const
- virtual os::smart_ptr< **streamCipher** > **buildStream** (unsigned char *data, uint32_t len) const
- virtual std::string **streamAlgorithmName** () const
- virtual uint16_t **streamAlgorithm** () const
- virtual std::string **hashAlgorithmName** () const
- virtual uint16_t **hashAlgorithm** () const
- void **setHashSize** (uint16_t **hashSize**)
- uint16_t **hashSize** () const

Protected Attributes

- uint16_t **_hashSize**

6.51.1 Constructor & Destructor Documentation

crypto::streamPackageFrame::streamPackageFrame (uint16_t hashSize = `size::hash256`) `[inline]`

virtual crypto::streamPackageFrame::~streamPackageFrame ( ) `[inline]`, `[virtual]`

6.51.2 Member Function Documentation

virtual os::smart_ptr<**streamCipher**> crypto::streamPackageFrame::buildStream (unsigned char * data, uint32_t len) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual os::smart_ptr<**streamPackageFrame**> crypto::streamPackageFrame::getCopy ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual uint16_t crypto::streamPackageFrame::hashAlgorithm ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual std::string crypto::streamPackageFrame::hashAlgorithmName ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual **hash** crypto::streamPackageFrame::hashCopy (unsigned char * data) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual **hash** crypto::streamPackageFrame::hashData (unsigned char * data, uint32_t len) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

virtual **hash** crypto::streamPackageFrame::hashEmpty ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 248).

uint16_t crypto::streamPackageFrame::hashSize ( ) const `[inline]`

void crypto::streamPackageFrame::setHashSize (uint16_t hashSize) `[inline]`

virtual uint16_t crypto::streamPackageFrame::streamAlgorithm ( ) const `[inline]`, `[virtual]`

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 249).

virtual std::string crypto::streamPackageFrame::streamAlgorithmName ( ) const [inline], [virtual]

Reimplemented in **crypto::streamPackage**< **streamType, hashType** > (p. 249).

6.51.3 Member Data Documentation

uint16_t crypto::streamPackageFrame::_hashSize [protected]

6.52 crypto::streamPackageTypeBank Class Reference

Public Member Functions

- virtual ~**streamPackageTypeBank** ()
- void **setDefaultPackage** (os::smart_ptr< **streamPackageFrame** > package)
- const os::smart_ptr< **streamPackageFrame** > **defaultPackage** () const
- void **pushPackage** (os::smart_ptr< **streamPackageFrame** > package)
- const os::smart_ptr< **streamPackageFrame** > **findStream** (uint16_t streamID, uint16_t hash↩ ID) const
- const os::smart_ptr< **streamPackageFrame** > **findStream** (const std::string &streamName, const std::string &hashName) const

Static Public Member Functions

- static os::smart_ptr< **streamPackageTypeBank** > **singleton** ()

Private Member Functions

- **streamPackageTypeBank** ()

Private Attributes

- os::smart_ptr< **streamPackageFrame** > **_defaultPackage**
- std::vector< os::smart_ptr< std::vector< os::smart_ptr< **streamPackageFrame** > > > > **package↩ Vector**

6.52.1 Constructor & Destructor Documentation

crypto::streamPackageTypeBank::streamPackageTypeBank ( ) [private]

virtual crypto::streamPackageTypeBank::~streamPackageTypeBank ( ) [inline], [virtual]

6.52.2 Member Function Documentation

const os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::defaultPackage ( ) const [inline]

const os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::findStream (uint16_t streamID, uint16_t hashID) const

const os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::findStream (const std::string & streamName, const std::string & hashName) const

void crypto::streamPackageTypeBank::pushPackage (os::smart_ptr< **streamPackageFrame** > package)

void crypto::streamPackageTypeBank::setDefaultPackage (os::smart_ptr< **streamPackageFrame** > package)

static os::smart_ptr<**streamPackageTypeBank**> crypto::streamPackageTypeBank::singleton ( ) `[static]`

6.52.3 Member Data Documentation

os::smart_ptr<**streamPackageFrame**> crypto::streamPackageTypeBank::_defaultPackage `[private]`

std::vector<os::smart_ptr<std::vector<os::smart_ptr<**streamPackageFrame**> > > > crypto::streamPackageTypeBank::packageVector `[private]`

6.53 crypto::streamPacket Class Reference

Public Member Functions

- **streamPacket** (os::smart_ptr< **streamCipher** > source, unsigned int s)
- virtual ~**streamPacket** ()
- uint16_t **getIdentifier** () const
- const uint8_t * **getPacket** () const
- uint8_t * **encrypt** (uint8_t *pt, unsigned int len, bool surpress=true) const

Private Attributes

- uint8_t * **packetArray**
- uint16_t **identifier**
- unsigned int **size**

6.53.1 Constructor & Destructor Documentation

crypto::streamPacket::streamPacket (os::smart_ptr< **streamCipher** > source, unsigned int s)

virtual crypto::streamPacket::~streamPacket ( ) `[virtual]`

6.53.2 Member Function Documentation

uint8_t* crypto::streamPacket::encrypt (uint8_t * pt, unsigned int len, bool surpress = `true`) const

uint16_t crypto::streamPacket::getIdentifier ( ) const

const uint8_t* crypto::streamPacket::getPacket ( ) const

6.53.3 Member Data Documentation

uint16_t crypto::streamPacket::identifier [private]

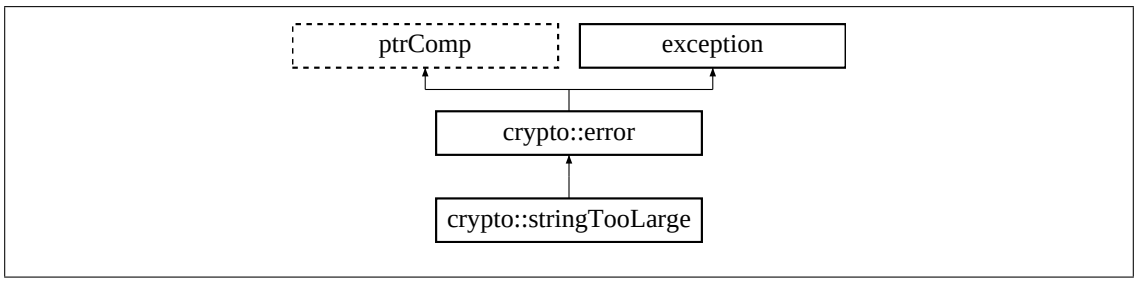
uint8_t* crypto::streamPacket::packetArray [private]

unsigned int crypto::streamPacket::size [private]

6.54 crypto::stringTooLarge Class Reference

String size error.

Inheritance diagram for crypto::stringTooLarge:



Public Member Functions

- virtual ~**stringTooLarge** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Group ID/Name Size Error".
- std::string **errorDescription** () const

  Long error descriptor Returns "Group ID or Name was larger than the maximum size. Please user a smaller string".

6.54.1 Detailed Description

String size error.

Thrown when either the username or group ID are too large.

6.54.2 Constructor & Destructor Documentation

virtual crypto::stringTooLarge::~stringTooLarge ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.54.3 Member Function Documentation

std::string crypto::stringTooLarge::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Group ID or Name was larger than the maximum size. Please user a smaller string".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::stringTooLarge::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Group ID/Name Size Error".

Returns

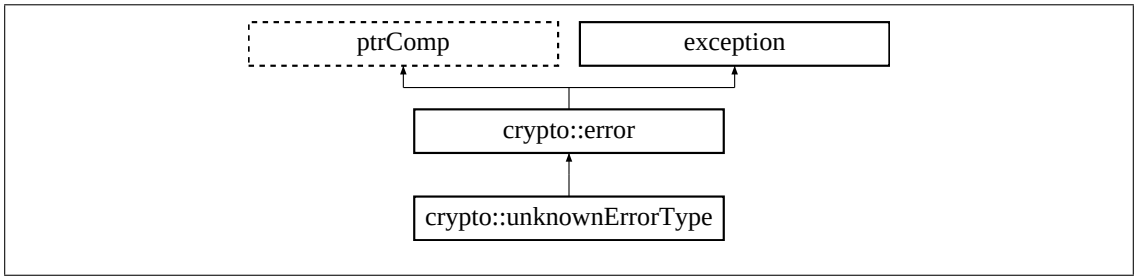
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.55 crypto::unknownErrorType Class Reference

Unknown error.

Inheritance diagram for crypto::unknownErrorType:



Public Member Functions

- virtual ~**unknownErrorType** () throw ()

  Virtual destructor.
- std::string **errorTitle** () const

  Short error descriptor Returns "Unknown Error Type".
- std::string **errorDescription** () const

  Long error descriptor Returns "Caught some exception, but the type is unknown".

6.55.1 Detailed Description

Unknown error.

Thrown when an error of undefined type occurs. Used as a catch-all exception.

6.55.2 Constructor & Destructor Documentation

virtual crypto::unknownErrorType::~unknownErrorType ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. Must explicitly declare that this function does not throw exceptions.

6.55.3 Member Function Documentation

std::string crypto::unknownErrorType::errorDescription ( ) const [inline], [virtual]

Long error descriptor Returns "Caught some exception, but the type is unknown".

Returns

Error description std::string

Reimplemented from **crypto::error** (p. 83).

std::string crypto::unknownErrorType::errorTitle ( ) const [inline], [virtual]

Short error descriptor Returns "Unknown Error Type".

Returns

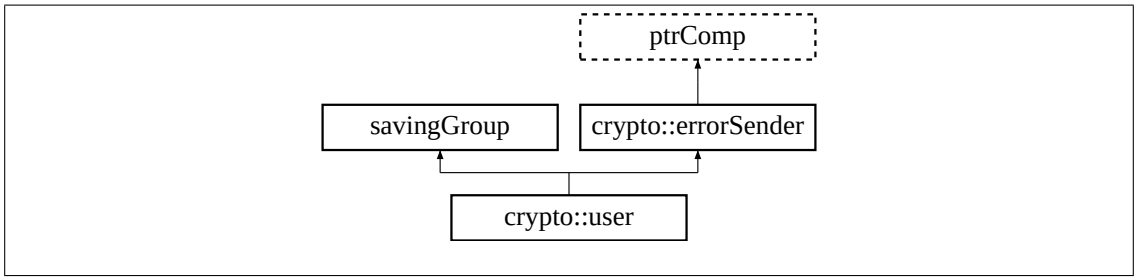
Error title std::string

Reimplemented from **crypto::error** (p. 84).

6.56 crypto::user Class Reference

Primary user class.

Inheritance diagram for crypto::user:



Public Member Functions

- bool **wasConstructed** () const

  Returns the construction state of the user.
- **user** (std::string **username**, std::string saveDir="", const unsigned char *key=NULL, unsigned int keyLen=0)

Constructs the user from scratch or directory.

- virtual ~**user** ()

  Virtual destructor.
- void **save** ()

  Saves all dependencies.
- void **setPassword** (const unsigned char *key=NULL, unsigned int keyLen=0)

  Set password.
- void **setStreamPackage** (os::smart_ptr< **streamPackageFrame** > strmPack)

  Set stream package.
- bool **setDefaultPublicKey** (os::smart_ptr< **publicKey** > key)

  Sets the default public key.
- bool **addPublicKey** (os::smart_ptr< **publicKey** > key)

  Attempt to add new public key.
- os::smart_ptr< **publicKey** > **findPublicKey** (os::smart_ptr< **publicKeyPackageFrame** > pkfrm)

  Find public key by information.
- unsigned char * **unsignedIDMessage** (unsigned int &len, std::string groupID="default", std::string nodeName="")

  Produces an unsigned ID message.
- bool **processIDMessage** (unsigned char *mess, unsigned int len)

  Process ID message.
- unsigned char * **encryptMessage** (unsigned int &finishedLen, const unsigned char *mess, unsigned int len, std::string groupID, std::string nodeName)

  Encrypt an out-going message.
- unsigned char * **decryptMessage** (unsigned int &finishedLen, const unsigned char *mess, unsigned int len, std::string groupID, std::string nodeName)

  Decrypt a message.
- const std::string & **username** () const

  Access name of user.
- const unsigned char * **password** () const

  Access raw password.
- unsigned int **passwordLength** () const

  Access password length.
- std::string **directory** () const

  Access save directory.
- os::smart_ptr< **streamPackageFrame** > **streamPackage** () const

  Access streaming package.
- os::smart_ptr< **keyBank** > **getKeyBank** ()

  Access key bank.
- os::smart_ptr< **publicKey** > **getDefaultPublicKey** ()

  Returns the default public key.
- os::smart_ptr< os::adnode< **publicKey** > > **getFirstPublicKey** ()

  Returns the first public key group.

- os::smart_ptr< os::adnode< **publicKey** > > **getLastPublicKey** ()

  Returns the last public key group.
- os::smart_ptr< **gatewaySettings** > **findSettings** (std::string group="default")

  Find gateway settings.
- os::smart_ptr< **gatewaySettings** > **insertSettings** (std::string group)

  Insert gateway settings.
- os::smart_ptr< os::adnode< **gatewaySettings** > > **getFirstSettings** ()

  Returns the first gateway settings group.
- os::smart_ptr< os::adnode< **gatewaySettings** > > **getLastSettings** ()

  Returns the last gateway settings group.
- os::smart_ptr< **publicKey** > **searchKey** (**hash** hsh, unsigned int &hist, bool &type)

  Searches for key by hash.
- os::smart_ptr< **publicKey** > **searchKey** (os::smart_ptr< **number** > key, unsigned int &hist, bool &type)

  Searches for key.
- os::smart_ptr< **publicKey** > **searchKey** (**hash** hsh)

  Searches for key.
- os::smart_ptr< **publicKey** > **searchKey** (os::smart_ptr< **number** > key)

  Searches for key.

Static Public Member Functions

- static bool **isIDMessage** (unsigned char m)

  Check if a message is an ID message.
- static bool **isDataMessage** (unsigned char m)

  Check if a message is a data message.
- static bool **isEncrypted** (unsigned char m)

  Check if a message is encrypted.

Protected Member Functions

- os::smartXMLNode **generateSaveTree** ()

  Creates meta-data XML file.

Protected Attributes

- bool **_wasConstructed**
- std::string **_username**

  Name of user.
- unsigned char * **_password**

  Primary symmetric key.
- unsigned int **_passwordLength**

  Length of symmetric key.
- std::string **_saveDir**

Save directory for user.

- os::smart_ptr< **streamPackageFrame** > **_streamPackage**

  Default stream package.
- os::smart_ptr< **keyBank** > **_keyBank**

  Key bank.
- os::asyncAVLTree< **publicKey** > **_publicKeys**

  Public keys.
- os::smart_ptr< **publicKey** > **_defaultKey**

  Default public key.
- os::asyncAVLTree< **gatewaySettings** > **_settings**

  List of gateway settings.

6.56.1 Detailed Description

Primary user class.

The user class defines a set of keys associated with a local user. This class notifies a set of listeners when various passwords and keys are changed, as this class allows for the encryption of a group of files with the provided keys

6.56.2 Constructor & Destructor Documentation

crypto::user::user (std::string username, std::string saveDir = "", const unsigned char * key = `NULL`, unsigned int keyLen = `0`)

Constructs the user from scratch or directory.

Constructs a user from a directory or from scratch. If the specified directory does not exists, this class creates the directory and begins to populate it. If no key is specified, all files are un-encrypted. If a key is specified, all files are encrypted with this key.

Parameters

`in`	*username*	Name of user to be saved
`in`	*saveDir*	Directory to save users in
`in`	*key*	Symetric key
`in`	*keyLen*	Length of symetric key

virtual crypto::user::~user ( ) `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

6.56.3 Member Function Documentation

bool crypto::user::addPublicKey (os::smart_ptr< **publicKey** > key)

Attempt to add new public key.

Attempts to add a public key to the public key bank. If successful, and if the default key is NULL, the added key becomes the default key.

Parameters

in	*key*	Public key to be added

Returns

True if successfully added, else, false

unsigned char* crypto::user::decryptMessage (unsigned int & finishedLen, const unsigned char * mess, unsigned int len, std::string groupID, std::string nodeName)

Decrypt a message.

Takes an array of data representing an encrypted message targeted for this user. The message is decrypted and returned.

Parameters

out	*finishedLen*	Length of the finished message
in	*mess*	Message to be decrypted
in	*len*	Length of the message to be decrypted
in	*groupID*	Group ID of message source
in	*nodeName*	Name of message source

Returns

Decrypted message

std::string crypto::user::directory ( ) const [inline]

Access save directory.

Returns

crypto::user::_saveDir (p. 267) + username

unsigned char* crypto::user::encryptMessage (unsigned int & finishedLen, const unsigned char * mess, unsigned int len, std::string groupID, std::string nodeName)

Encrypt an out-going message.

Takes an array of data and encrypts it with the default public-key of the target user. Takes a group ID and node name to target the message.

Parameters

out	*finishedLen*	Length of the finished message
in	*mess*	Message to be encrypted
in	*len*	Length of message to be encrypted
in	*groupID*	String of the target group
in	*nodeName*	String of the name of the target node

Returns

Encrypted message pointer

os::smart_ptr<**publicKey**> crypto::user::findPublicKey (os::smart_ptr< **publicKeyPackageFrame** > pkfrm)

Find public key by information.

Searches for a public key with the given' characteristics. Keys are searched by algorithm and size.

Parameters

in	*pkfrm*	Public key information to match

Returns

Public key matching intrinsics

os::smart_ptr<**gatewaySettings**> crypto::user::findSettings (std::string group = "default")

Find gateway settings.

Parameters

in	*group*	Name of group of the settings

Returns

Pointer to the found gateway settings

os::smartXMLNode crypto::user::generateSaveTree ( ) [protected]

Creates meta-data XML file.

Constructs and returns the XML tree for this class. The XML tree may or may not be encrypted.

Returns

XML tree for saving

os::smart_ptr<**publicKey**> crypto::user::getDefaultPublicKey ( ) [inline]

Returns the default public key.

Returns

crypto::user::_defaultKey (p. 267)

os::smart_ptr<os::adnode<**publicKey**> > crypto::user::getFirstPublicKey ( ) [inline]

Returns the first public key group.
Allows programs to list off the available key groups bound to this user

Returns

crypto::user::_publicKeys.getFirst()

os::smart_ptr<os::adnode<**gatewaySettings**> > crypto::user::getFirstSettings ( ) [inline]

Returns the first gateway settings group.
Allows programs to list off the available gateway settings bound to this user

Returns

crypto::user::_settings.getFirst()

os::smart_ptr<**keyBank**> crypto::user::getKeyBank ( ) [inline]

Access key bank.

Returns

crypto::user::_keyBank (p. 267)

os::smart_ptr<os::adnode<**publicKey**> > crypto::user::getLastPublicKey ( ) [inline]

Returns the last public key group.
Allows programs to list off the available key groups bound to this user

Returns

crypto::user::_publicKeys.getFirst()

os::smart_ptr<os::adnode<**gatewaySettings**> > crypto::user::getLastSettings ( ) [inline]

Returns the last gateway settings group.
Allows programs to list off the available gateway settings bound to this user

Returns

crypto::user::_settings.getLast()

os::smart_ptr<**gatewaySettings**> crypto::user::insertSettings (std::string group)

Insert gateway settings.

Parameters

in	*group*	Name of group of the settings

Returns

Point to the inserted gateway settings

static bool crypto::user::isDataMessage (unsigned char m) [inline], [static]

Check if a message is a data message.
Checks the first byte of a message to see if it is a data message.

Returns

True if a data message, else, false

static bool crypto::user::isEncrypted (unsigned char m) [inline], [static]

Check if a message is encrypted.
Checks the first byte of a message to see if it is encrypted

Returns

True if encrypted, else, false

static bool crypto::user::isIDMessage (unsigned char m) [inline], [static]

Check if a message is an ID message.
Checks the first byte of a message to see if it is an ID message.

Returns

True if an ID message, else, false

const unsigned char* crypto::user::password ( ) const [inline]

Access raw password.

Returns

crypto::user::_password (p. 267)

unsigned int crypto::user::passwordLength ( ) const [inline]

Access password length.

Returns

crypto::user::_passwordLength (p. 267)

bool crypto::user::processIDMessage (unsigned char * mess, unsigned int len)

Process ID message.

Processes any ID message. Note that this function can process both targeted and non-targeted ID messages.

Parameters

in	*mess*	Incoming message
in	*len*	Length of incoming message

Returns

True if valid ID message, else, false

void crypto::user::save ( )

Saves all dependencies.
This function saves all dependencies based on the save queue.

Returns

void

os::smart_ptr<**publicKey**> crypto::user::searchKey (**hash** hsh, unsigned int & hist, bool & type)

Searches for key by hash.
Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*hsh*	Hash of the key to be searched for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (os::smart_ptr< **number** > key, unsigned int & hist, bool & type)

Searches for key.
Binds the location that the keys were found in to the arguments of the function.

Parameters

in	*num*	Key to search for
out	*hist*	History value the key was found
out	*type*	Type (public or private)

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (**hash** hsh) `[inline]`

Searches for key.

Parameters

`in`	*num*	Key to search for

Returns

Key pair conatining the searched key

os::smart_ptr<**publicKey**> crypto::user::searchKey (os::smart_ptr< **number** > key) `[inline]`

Searches for key.

Parameters

`in`	*num*	Key to search for

Returns

Key pair conatining the searched key

bool crypto::user::setDefaultPublicKey (os::smart_ptr< **publicKey** > key)

Sets the default public key.

Attempts to bind a public key as the default public key. First checks if the key in question exists and binds the key with the characteristics of the provided key as the default key.

Parameters

`in`	*key*	Public key to be bound as the default key

Returns

True if default key bound, else, false

void crypto::user::setPassword (const unsigned char * key = `NULL`, unsigned int keyLen = `0`)

Set password.

Sets symetric key used to securely save user data.

Parameters

in	*key*	Symetric key
in	*keyLen*	Length of symetric key

Returns

void

void crypto::user::setStreamPackage (os::smart_ptr< **streamPackageFrame** > strmPack)

Set stream package.

Binds a new stream package. Calls for saving of this user.

Parameters

in	*strmPack*	Stream package

Returns

void

os::smart_ptr<**streamPackageFrame**> crypto::user::streamPackage ( ) const [inline]

Access streaming package.

Returns

crypto::user::_streamPackage (p. 267)

unsigned char* crypto::user::unsignedIDMessage (unsigned int & len, std::string groupID = "default", std::string nodeName = "")

Produces an unsigned ID message.

Generates an identification message to be sent to a node. If the target node is specified, this function will encrypt the target message for that target node.

Parameters

out	*len*	Length of returned array
in	*groupID*	Group this user is part of
in	*nodeName*	Name of target node

Returns

Unsigned ID message

const std::string& crypto::user::username ( ) const [inline]

Access name of user.

Returns

crypto::user::_username (p. 268)

bool crypto::user::wasConstructed ( ) const [inline]

Returns the construction state of the user.

Returns

crypto::bool::_wasConstructed

6.56.4 Member Data Documentation

os::smart_ptr<**publicKey**> crypto::user::_defaultKey [protected]

Default public key.

Sets the default public key definition. Note that a default public key will be defined the moment any public key is bound to a user.

os::smart_ptr<**keyBank**> crypto::user::_keyBank [protected]

Key bank.

This key bank defines all of the public keys which are known by this user

unsigned char* crypto::user::_password [protected]

Primary symmetric key.

unsigned int crypto::user::_passwordLength [protected]

Length of symmetric key.

os::asyncAVLTree<**publicKey**> crypto::user::_publicKeys [protected]

Public keys.

This stores all public keys accociated with this specific user.

std::string crypto::user::_saveDir [protected]

Save directory for user.

os::asyncAVLTree<**gatewaySettings**> crypto::user::_settings [protected]

List of gateway settings.

os::smart_ptr<**streamPackageFrame**> crypto::user::_streamPackage [protected]

Default stream package.

std::string crypto::user::_username [protected]

Name of user.

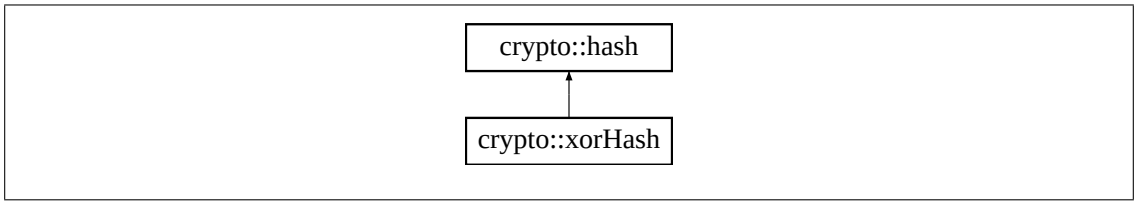
bool crypto::user::_wasConstructed [protected]

Stores if the user was constructed

6.57 crypto::xorHash Class Reference

XOR hash class.

Inheritance diagram for crypto::xorHash:



Public Member Functions

- **xorHash** ()

  Default XOR hash constructor.
- **xorHash** (const unsigned char ∗**data**, uint16_t **size**)

  Raw data copy.
- **xorHash** (const **xorHash** &cpy)

  XOR copy constructor.
- void **preformHash** (const unsigned char ∗**data**, uint32_t dLen)

  Binds a data-set.
- std::string **algorithmName** () const

  Algorithm name string access.

Static Public Member Functions

- static std::string **staticAlgorithmName** ()

  Algorithm name string access.
- static uint16_t **staticAlgorithm** ()

  Algorithm ID number access.
- static **xorHash hash64Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 64 bit hash.
- static **xorHash hash128Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 128 bit hash.
- static **xorHash hash256Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 256 bit hash.

- static **xorHash hash512Bit** (const unsigned char ∗**data**, uint32_t length)

  Static 512 bit hash.

Private Member Functions

- **xorHash** (const unsigned char ∗**data**, uint32_t length, uint16_t **size**)

  XOR hash constructor.

Additional Inherited Members

6.57.1 Detailed Description

XOR hash class.

This class defines an XOR based hash. Note that this hash is not cryptographically secure and essentially just acts as a checksum.

6.57.2 Constructor & Destructor Documentation

crypto::xorHash::xorHash (const unsigned char ∗ data, uint32_t length, uint16_t size) `[private]`

XOR hash constructor.

Constructs a hash with the data to be hashed, the length of the array and the size of the hash to be constructed.

Parameters

`in`	*data*	Data array
`in`	*length*	Length of data array
`in`	*size*	Size of hash

crypto::xorHash::xorHash ( ) `[inline]`

Default XOR hash constructor.

Constructs an empty XOR hash class.

crypto::xorHash::xorHash (const unsigned char ∗ data, uint16_t size)

Raw data copy.

Initializes the XOR hash with a data array. This data array is not hashed but assumed to represent hashed data.

Parameters

`in`	*data*	Hashed data array
`in`	*size*	Size of hash array

crypto::xorHash::xorHash (const **xorHash** & cpy) [inline]

XOR copy constructor.
Constructs an XOR hash with another XOR hash.

Parameters

in	*cpy*	Hash to be copied

6.57.3 Member Function Documentation

std::string crypto::xorHash::algorithmName ( ) const [inline], [virtual]

Algorithm name string access.
Returns the name of the current algorithm string. This function requires an instantiated XOR hash.

Returns

"XOR"

Reimplemented from **crypto::hash** (p. 118).

static **xorHash** crypto::xorHash::hash128Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 128 bit hash.
Hashes the provided data array with the XOR algorithm, returning a 128 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **xorHash** crypto::xorHash::hash256Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 256 bit hash.
Hashes the provided data array with the XOR algorithm, returning a 256 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **xorHash** crypto::xorHash::hash512Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 512 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 512 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

static **xorHash** crypto::xorHash::hash64Bit (const unsigned char * data, uint32_t length) [inline], [static]

Static 64 bit hash.

Hashes the provided data array with the XOR algorithm, returning a 64 bit XOR hash.

Parameters

data	Data array to be hashed
length	Length of data array to be hashed

Returns

New **xorHash** (p. 268)

void crypto::xorHash::preformHash (const unsigned char * data, uint32_t dLen)

Binds a data-set.

Preforms the hash algorithm on the set of data provided and binds the result to this hash.

Parameters

in	*data*	Data array to be hashed
in	*dLen*	Length of data array

static uint16_t crypto::xorHash::staticAlgorithm ( ) [inline], [static]

Algorithm ID number access.

Returns the ID of the current algorithm. This function is static and can be accessed without instantiating the class.

Returns

crypto::algo::hashXOR

static std::string crypto::xorHash::staticAlgorithmName ( ) [inline], [static]

Algorithm name string access.

Returns the name of the current algorithm string. This function is static and can be accessed without instantiating the class.

Returns

"XOR"

Part II

Datastructures Library

Chapter 7

Introduction

The Datastructures library contains a series of utility classes and template classes used for the organization and management of data. Most notably, this library allow dynamic memory management through the smart_ptr class and provides a flexible runtime data container in the ads (Abstract Data Structure) template and its children.

7.1 Unit Testing

The testing of the Datastructures library is preformed within the UnitTest library. Since the UnitTest library uses the functionality of the Datastructures library, the Datastructures library cannot be dependent on the UnitTest library as the UnitTest library is already dependent on the Datastructures library

7.2 Namespace os

Datastructures extends the os namespace. The os namespace is designed for tools, algorithms and data-structures used in programs of all types. Structures in this library do not implement operating system specific interfaces such as sockets and file I/O. The osMechanics library also extends the os namespace.

Chapter 8

File Index

8.1 File List

Here is a list of all files with brief descriptions:

staticConstantPrinter.h

Chapter 9

File Documentation

9.1 Datastructures.h File Reference

Master Datastructures header file.

9.1.1 Detailed Description

Master Datastructures header file.

Author

Jonathan Bedard

Date

2/14/2016

Bug No known bugs.

All of the headers in the Datastructures library are held in this file. When using the Datastructures library, it is expected that this header is included instead of the individual required headers.

9.2 abstractSorting.h File Reference

Template for sorting arrays.

Namespaces

- **os**

Functions

- template<class dataType >
  int **os::defaultCompareSort** (const dataType &v1, const dataType &v2)

  Basic compare.

- template<class dataType >
  int **os::pointerCompareSort** (smart_ptr< dataType > ptr1, smart_ptr< dataType > ptr2)

  Raw pointer compare.
- template<class dataType >
  void **os::quicksort** (dataType *arr, unsigned int length, int(*sort_comparison)(const dataType &, const dataType &)=&defaultCompareSort)

  Template quick-sort.
- template<class dataType >
  void **os::pointerQuicksort** (smart_ptr< smart_ptr< dataType > > arr, unsigned int length, int(*sort_comparison)(smart_ptr< dataType >, smart_ptr< dataType >)=&pointerCompare↩ Sort)

  Template for quick-sort, pointer version.

9.2.1 Detailed Description

Template for sorting arrays.

Author

Jonathan Bedard

Date

2/15/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is thread safe.

9.3 ads.h File Reference

Abstract datastructure interface.

Classes

- class **os::ptrComp**

  Pointer compare interface.
- class **os::adnode**< **dataType** >

  Abstract data-node.
- class **os::ads**< **dataType** >

  Abstract datastructure.

Namespaces

- **os**

9.3.1 Detailed Description

Abstract datastructure interface.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains definitions of a set of class interfaces used by abstract datastructures and classes interfacing with abstract datastructures.

9.4 asyncAVL.h File Reference

Asynchronous AVL tree.

Classes

- class **os::asyncAVLNode**< **dataType** >

  Node for usage in an asynchronous AVL tree.
- class **os::asyncAVLTree**< **dataType** >

  Asynchronous balanced binary search tree.

Namespaces

- **os**

9.4.1 Detailed Description

Asynchronous AVL tree.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is thread safe.

9.5 AVL.h File Reference

AVL tree.

Classes

- class **os::AVLNode**< **dataType** >
  Node for usage in an AVL tree.
- class **os::AVLTree**< **dataType** >
  Balanced binary search tree.

Namespaces

- **os**

9.5.1 Detailed Description

AVL tree.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains a template class definition of an AVL tree and its nodes. This tree has insertion, search and deletion of O(log(n)) where n is the number of nodes in the tree. This tree is not thread safe.

9.6 eventDriver.h File Reference

Event sender and receiver.

Classes

- class **os::eventSender**< **receiverType** >
  Class which enables event sending.
- class **os::eventReceiver**< **senderType** >
  Class which enables event receiving.

Namespaces

- **os**

Variables

- std::recursive_mutex * **os::eventLock**
  Event processing mutex.

9.6.1 Detailed Description

Event sender and receiver.

Author

Jonathan Bedard

Date

5/9/2016

Bug No known bugs.

Both **os::eventReceiver** (p. 360) and **os::eventSender** (p. 363) are experimental classes and have not been tested or utilized.

9.7 eventDriver.cpp File Reference

Event driver implementation.

9.7.1 Detailed Description

Event driver implementation.

Author

Jonathan Bedard

Date

2/28/2016

Bug No known bugs.

This file implements **os::eventLock** (p. 321) for **os::eventSender** (p. 363) and **os::eventReceiver** (p. 360). These are experimental class and not yet used or tested

9.8 list.h File Reference

Doubly Linked List.

Classes

- class **os::unsortedListNode**< **dataType** >
  Node for usage in a linked list.
- class **os::unsortedList**< **dataType** >
  Unsorted linked list.

Namespaces

- **os**

9.8.1 Detailed Description

Doubly Linked List.

Author

Jonathan Bedard

Date

2/1/2016

Bug No known bugs.

This file contains a template class definition of a linked list and its nodes. This list has insertion, find and delete of O(n). The linked list provided is doubly linked, allowing for forward and backward traversal. This list is not thread safe.

9.9 matrix.h File Reference

Matrix templates.

Classes

- class **os::matrix**< **dataType** >

  Raw matrix.
- class **os::indirectMatrix**< **dataType** >

  Indirect matrix.

Namespaces

- **os**

Functions

- template<class dataType >
  bool **os::compareSize** (const matrix< dataType > &m1, const matrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **os::compareSize** (const indirectMatrix< dataType > &m1, const matrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **os::compareSize** (const matrix< dataType > &m1, const indirectMatrix< dataType > &m2)

Compares the size of two matrices.

- template<class dataType >
  bool **os::compareSize** (const indirectMatrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **os::testCross** (const matrix< dataType > &m1, const matrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **os::testCross** (const indirectMatrix< dataType > &m1, const matrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **os::testCross** (const matrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **os::testCross** (const indirectMatrix< dataType > &m1, const indirectMatrix< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **operator==** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for equality.
- template<class dataType >
  bool **operator==** (const **os::indirectMatrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for equality.
- template<class dataType >
  bool **operator==** (const **os::matrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Test for equality.
- template<class dataType >
  bool **operator==** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< data↩ Type > &m2)

  Test for equality.
- template<class dataType >
  bool **operator!=** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for inequality.
- template<class dataType >
  bool **operator!=** (const **os::indirectMatrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Test for inequality.
- template<class dataType >
  bool **operator!=** (const **os::matrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Test for inequality.

- template<class dataType >
  bool **operator!=** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< data↩ Type > &m2)

  Test for inequality.
- template<class dataType >
  os::matrix< dataType > **operator+** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Addition.
- template<class dataType >
  os::matrix< dataType > **operator+** (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Addition.
- template<class dataType >
  os::matrix< dataType > **operator+** (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Addition.
- template<class dataType >
  os::indirectMatrix< dataType > **operator+** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Addition.
- template<class dataType >
  os::matrix< dataType > **operator-** (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Subtraction.
- template<class dataType >
  os::matrix< dataType > **operator-** (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Subtraction.
- template<class dataType >
  os::matrix< dataType > **operator-** (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Subtraction.
- template<class dataType >
  os::indirectMatrix< dataType > **operator-** (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Subtraction.
- template<class dataType >
  os::matrix< dataType > **operator**∗ (const **os::matrix**< dataType > &m1, const **os::matrix**< dataType > &m2)

  Cross-product.
- template<class dataType >
  os::matrix< dataType > **operator**∗ (const **os::indirectMatrix**< dataType > &m1, const **os**↩ **::matrix**< dataType > &m2)

  Cross-product.

- template<class dataType >
  os::matrix< dataType > **operator**∗ (const **os::matrix**< dataType > &m1, const **os::indirect**↩ **Matrix**< dataType > &m2)

  Cross-product.
- template<class dataType >
  os::indirectMatrix< dataType > **operator**∗ (const **os::indirectMatrix**< dataType > &m1, const **os::indirectMatrix**< dataType > &m2)

  Cross-product.
- template<class dataType >
  os::matrix< dataType > **operator**∗ (const dataType &d1, const **os::matrix**< dataType > &m1)

  Scalar multiplication.
- template<class dataType >
  os::matrix< dataType > **operator**∗ (const **os::matrix**< dataType > &m1, const dataType &d1)

  Scalar multiplication.
- template<class dataType >
  os::matrix< dataType > **operator**/ (const **os::matrix**< dataType > &m1, const dataType &d1)

  Scalar division.
- template<class dataType >
  os::indirectMatrix< dataType > **operator**∗ (const dataType &d1, const **os::indirectMatrix**< dataType > &m1)

  Scalar multiplication.
- template<class dataType >
  os::indirectMatrix< dataType > **operator**∗ (const **os::indirectMatrix**< dataType > &m1, const dataType &d1)

  Scalar multiplication.
- template<class dataType >
  os::indirectMatrix< dataType > **operator**/ (const **os::indirectMatrix**< dataType > &m1, const dataType &d1)

  Scalar division.
- template<class dataType >
  std::ostream & **operator**<< (std::ostream &os, const **os::matrix**< dataType > &dt)

  Prints out a matrix.
- template<class dataType >
  std::ostream & **operator**<< (std::ostream &os, const **os::indirectMatrix**< dataType > &dt)

  Prints out a matrix.

9.9.1 Detailed Description

Matrix templates.

Author

Jonathan Bedard

Date

2/2/2016

Bug No known bugs.

This file contains two template class definitions for matrices. One of these is an "indirect" matrix, meaning that the is an array of pointers, and the other is a direct matrix, meaning the matrix is an array of values.

9.9.2 Function Documentation

template<class dataType > bool operator!= (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for inequality.
Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for inequality.
Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for inequality.
Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

False if exactly equivalent

template<class dataType > bool operator!= (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for inequality.

Calls '==' and then inverts the result. Depends on the '!=' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

False if exactly equivalent

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '∗' and '+=' operator of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '∗' and '+=' operator of the dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '∗' and '+=' operator of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 x m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator∗ (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Cross-product.

Preforms the cross-product. The cross- product is undefined if the width of m1 does not equal the height of m2. If the cross-product is undefined, a matrix of size (0,0) will be returned. Depends on the '∗' and '+=' operator of the dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 x m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator∗ (const dataType & d1, const **os::matrix**< dataType > & m1)

Scalar multiplication.

Multiplies a matrix by a constant. This function depends on the '∗' operator of the dataType.

Parameters

in	*d1*	Scalar data type
in	*m1*	Raw matrix reference

Returns

d1 ∗ m1 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator∗ (const **os::matrix**< dataType > & m1, const dataType & d1)

Scalar multiplication.

Multiplies a matrix by a constant. This function depends on the '∗' operator of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*d1*	Scalar data type

Returns

d1 ∗ m1 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator∗ (const dataType & d1, const **os::indirectMatrix**< dataType > & m1)

Scalar multiplication.

Multiplies an indirect matrix by a constant. This function depends on the '∗' operator of the data↩ Type.

Parameters

in	*d1*	Scalar data type
in	*m1*	Indirect matrix reference

Returns

d1 ∗ m1 (indirect matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator∗ (const **os::indirectMatrix**< dataType > & m1, const dataType & d1)

Scalar multiplication.

Multiplies an indirect matrix by a constant. This function depends on the '∗' operator of the data↩ Type.

Parameters

in	*m1*	Indirect matrix reference
in	*d1*	Scalar data type

Returns

d1 * m1 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator+ (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 + m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator+ (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Addition.

Preforms matrix addition. Matrix addition is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '+' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 + m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::matrix**<dataType> operator- (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 - m2 (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator- (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Subtraction.

Preforms matrix subtraction. Matrix subtraction is undefined if the two matrices are of different size. If the operation is undefined, a matrix of size (0,0) will be returned. Depends on the '-' operator of dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

m1 - m2 (indirect matrix)

template<class dataType > **os::matrix**<dataType> operator/ (const **os::matrix**< dataType > & m1, const dataType & d1)

Scalar division.

Divides a matrix by a constant. This function depends on the '/' operator of the dataType. No zero check, as the dataType is not defined.

Parameters

in	*m1*	Raw matrix reference
in	*d1*	Scalar data type

Returns

m1/d (raw matrix)

template<class dataType > **os::indirectMatrix**<dataType> operator/ (const **os::indirectMatrix**< dataType > & m1, const dataType & d1)

Scalar division.

Divides an indirect matrix by a constant. This function depends on the '/' operator of the dataType. No zero check, as the dataType is not defined.

Parameters

in	*m1*	Raw matrix reference
in	*d1*	Scalar data type

Returns

m1/d (raw matrix)

template<class dataType > std::ostream& operator<< (std::ostream & os, const **os::matrix**< dataType > & dt)

Prints out a matrix.

Prints out the entire matrix in the provided output stream. This matrix will be printed out in text form and requires the dataType of the matrix to define an ostream operator.

Parameters

	[in/out]	os std::ostream reference
in	*dt*	Raw matrix reference

Returns

std::ostream os

template<class dataType > std::ostream& operator<< (std::ostream & os, const **os::indirectMatrix**< dataType > & dt)

Prints out a matrix.

Prints out the entire matrix in the provided output stream. This matrix will be printed out in text form and requires the dataType of the matrix to define an ostream operator.

Parameters

	[in/out]	os std::ostream reference
in	*dt*	Indirect matrix reference

Returns

std::ostream os

template<class dataType > bool operator== (const **os::matrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::indirectMatrix**< dataType > & m1, const **os::matrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::matrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

True if exactly equivalent

template<class dataType > bool operator== (const **os::indirectMatrix**< dataType > & m1, const **os::indirectMatrix**< dataType > & m2)

Test for equality.

Tests the two matrices for equal size and then tests each matrix element for equality as well. This function is dependent on the '!=' definition of the dataType.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

True if exactly equivalent

9.10 osLogger.h File Reference

Logging for os namespace.

Namespaces

- **os**

Functions

- std::ostream & **os::osout_func** ()

  Standard out object for os namespace.

- std::ostream & **os::oserr_func** ()
  Standard error object for os namespace.

Variables

- smart_ptr< std::ostream > **os::osout_ptr**
  Standard out pointer for os namespace.
- smart_ptr< std::ostream > **os::oserr_ptr**
  Standard error pointer for os namespace.

9.10.1 Detailed Description

Logging for os namespace.
Jonathan Bedard

Date

1/30/2016

Bug No known bugs.

This file contains declarations which are used for logging within the os namespace.

9.11 osLogger.cpp File Reference

Logging for os namespace, implementation.

9.11.1 Detailed Description

Logging for os namespace, implementation.
Jonathan Bedard

Date

2/15/2016

Bug No known bugs.

This file contains global functions and variables used for logging in the os namespace.

9.12 osVectors.h File Reference

Vector templates.

Classes

- class **os::vector2d**< **dataType** >
  2-dimensional vector
- class **os::vector3d**< **dataType** >
  3-dimensional vector

Namespaces

- **os**

Typedefs

- typedef vector2d< int8_t > **os::vector2d_8**
  8 bit 2-d vector
- typedef vector2d< uint8_t > **os::vector2d_u8**
  unsigned 8 bit 2-d vector
- typedef vector2d< int16_t > **os::vector2d_16**
  16 bit 2-d vector
- typedef vector2d< uint16_t > **os::vector2d_u16**
  unsigned 16 bit 2-d vector
- typedef vector2d< int32_t > **os::vector2d_32**
  32 bit 2-d vector
- typedef vector2d< uint32_t > **os::vector2d_u32**
  unsigned 32 bit 2-d vector
- typedef vector2d< int64_t > **os::vector2d_64**
  64 bit 2-d vector
- typedef vector2d< uint64_t > **os::vector2d_u64**
  unsigned 64 bit 2-d vector
- typedef vector2d< float > **os::vector2d_f**
  float 2-d vector
- typedef vector2d< double > **os::vector2d_d**
  double 2-d vector
- typedef vector3d< int8_t > **os::vector3d_8**
  8 bit 3-d vector
- typedef vector3d< uint8_t > **os::vector3d_u8**
  unsigned 8 bit 3-d vector
- typedef vector3d< int16_t > **os::vector3d_16**
  16 bit 3-d vector
- typedef vector3d< uint16_t > **os::vector3d_u16**
  unsigned 16 bit 3-d vector
- typedef vector3d< int32_t > **os::vector3d_32**
  32 bit 3-d vector
- typedef vector3d< uint32_t > **os::vector3d_u32**
  unsigned 32 bit 3-d vector
- typedef vector3d< int64_t > **os::vector3d_64**
  64 bit 3-d vector
- typedef vector3d< uint64_t > **os::vector3d_u64**
  unsigned 64 bit 3-d vector
- typedef vector3d< float > **os::vector3d_f**
  float 3-d vector
- typedef vector3d< double > **os::vector3d_d**
  double 3-d vector

9.12.1 Detailed Description

Vector templates.

Author

Jonathan Bedard

Date

3/12/2016

Bug No known bugs.

This file contains two template classes defining vector objects. Vectors can, in a broad sense, be used for any class which defines general mathematical operations. This particular file offers vector type definitions for all of the basic integer and floating point types.

9.13 set.h File Reference

Smart Set.

Classes

- class **os::smartSet**< **dataType** >
  Smart set abstract data-structures.

Namespaces

- **os**

Enumerations

- enum **os::setTypes** { **os::def_set** =0, **os::small_set**, **os::sorted_set** }
  Index of abstract data-structures.

9.13.1 Detailed Description

Smart Set.

Author

Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains a template class defining a "smart set." A smart set wraps other forms of abstract data structures, allowing applications to define abstract data-structures by numbered indexes.

9.14 smartPointer.h File Reference

Template declaration of **os::smart_ptr** (p. 378).

Classes

- class **os::smart_ptr**< **dataType** >
  Reference counted pointer.

Namespaces

- **os**

Typedefs

- typedef void(∗ **os::void_rec**) (void ∗)
  Deletion function typedef.

Enumerations

- enum **os::smart_pointer_type** {
  os::null_type =0, **os::raw_type**, **os::shared_type**, **os::shared_type_array**,
  os::shared_type_dynamic_delete }
  *Enumeration for types of **os::smart_ptr** (p. 378).*

Functions

- template<class targ , class src >
  smart_ptr< targ > **os::cast** (const **os::smart_ptr**< src > &conv)
  ***os::smart_ptr** (p. 378) cast function*
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >
  bool **operator==** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >
  bool **operator==** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator==** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const long c2)

- template<class dataType >
  bool **operator==** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator==** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >
  bool **operator==** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >
  bool **operator!=** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >
  bool **operator!=** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator!=** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >
  bool **operator!=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator!=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >
  bool **operator!=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >
  bool **operator**< (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >
  bool **operator**< (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator**< (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >
  bool **operator**< (const long c1, const **os::smart_ptr**< dataType > &c2)

- template<class dataType >
  bool **operator**< (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >
  bool **operator**< (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const dataType ∗c2)
- template<class dataType >
  bool **operator**<**=** (const dataType ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >
  bool **operator**<**=** (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator**<**=** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >
  bool **operator**<**=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**<**=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >
  bool **operator**<**=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const dataType ∗&c2)
- template<class dataType >
  bool **operator**> (const dataType ∗&c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const void ∗c2)
- template<class dataType >
  bool **operator**> (const void ∗c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator**> (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >
  bool **operator**> (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**> (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)

- template<class dataType >
  bool **operator**> (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const dataType *&c2)
- template<class dataType >
  bool **operator**>**=** (const dataType *&c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const void *c2)
- template<class dataType >
  bool **operator**>**=** (const void *c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const int c2)
- template<class dataType >
  bool **operator**>**=** (const int c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const long c2)
- template<class dataType >
  bool **operator**>**=** (const long c1, const **os::smart_ptr**< dataType > &c2)
- template<class dataType >
  bool **operator**>**=** (const **os::smart_ptr**< dataType > &c1, const unsigned long c2)
- template<class dataType >
  bool **operator**>**=** (const unsigned long c1, const **os::smart_ptr**< dataType > &c2)

9.14.1 Detailed Description

Template declaration of **os::smart_ptr** (p. 378).

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This file contains a template declaration of **os::smart_ptr** (p. 378) and supporting constants and functions. Note that because **os::smart_ptr** (p. 378) is a template class, the implimentation of **os::smart_ptr** (p. 378) occurs here as well.

9.14.2 Function Documentation

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) `[inline]`

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const dataType * c2) [inline]

template<class dataType > bool operator!= (const dataType * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator!= (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator!= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator!= (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator!= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator!= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const dataType * c2) [inline]

template<class dataType > bool operator< (const dataType * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator< (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator< (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator< (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator< (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator< (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const dataType * c2) [inline]

template<class dataType > bool operator<= (const dataType * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator<= (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator<= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator<= (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator<= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator<= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const dataType * c2) [inline]

template<class dataType > bool operator== (const dataType * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator== (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator== (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator== (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator== (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator== (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const dataType *& c2) [inline]

template<class dataType > bool operator> (const dataType *& c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator> (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator> (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator> (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator> (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator> (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const dataType *& c2) [inline]

template<class dataType > bool operator>= (const dataType *& c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const void * c2) [inline]

template<class dataType > bool operator>= (const void * c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const int c2) [inline]

template<class dataType > bool operator>= (const int c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const long c2) [inline]

template<class dataType > bool operator>= (const long c1, const **os::smart_ptr**< dataType > & c2) [inline]

template<class dataType > bool operator>= (const **os::smart_ptr**< dataType > & c1, const unsigned long c2) [inline]

template<class dataType > bool operator>= (const unsigned long c1, const **os::smart_ptr**< dataType > & c2) [inline]

9.15 staticConstantPrinter.h File Reference

Constant printing support.

Classes

- class **os::constantPrinter**

  Prints constant arrays to files.

Namespaces

- **os**

9.15.1 Detailed Description

Constant printing support.

Author

Jonathan Bedard

Date

1/31/2016

Bug No known bugs.

This file contains a class which helps facilitate printing massive tables of constants. It outputs .h and .cpp files with configured arrays of constants.

9.16 staticConstantPrinter.cpp File Reference

Constant printing support, implementation.

9.16.1 Detailed Description

Constant printing support, implementation.

Author

Jonathan Bedard

Date

4/618/2016

Bug No known bugs.

This file implements **os::constantPrinter** (p. 356). Consult **staticConstantPrinter.h** (p. 306) for detailed documentation.

Chapter 10

Class Index

10.1 Class List

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

Chapter 11

Namespace Documentation

11.1 os Namespace Reference

Classes

- class **adnode**

  Abstract data-node.
- class **ads**

  Abstract datastructure.
- class **asyncAVLNode**

  Node for usage in an asynchronous AVL tree.
- class **asyncAVLTree**

  Asynchronous balanced binary search tree.
- class **AVLNode**

  Node for usage in an AVL tree.
- class **AVLTree**

  Balanced binary search tree.
- class **constantPrinter**

  Prints constant arrays to files.
- class **eventReceiver**

  Class which enables event receiving.
- class **eventSender**

  Class which enables event sending.
- class **indirectMatrix**

  Indirect matrix.
- class **matrix**

  Raw matrix.
- class **ptrComp**

  Pointer compare interface.
- class **smart_ptr**

  Reference counted pointer.

- class **smartSet**

  Smart set abstract data-structures.
- class **unsortedList**

  Unsorted linked list.
- class **unsortedListNode**

  Node for usage in a linked list.
- class **vector2d**

  2-dimensional vector
- class **vector3d**

  3-dimensional vector

Typedefs

- typedef **vector2d**< int8_t > **vector2d_8**

  8 bit 2-d vector
- typedef **vector2d**< uint8_t > **vector2d_u8**

  unsigned 8 bit 2-d vector
- typedef **vector2d**< int16_t > **vector2d_16**

  16 bit 2-d vector
- typedef **vector2d**< uint16_t > **vector2d_u16**

  unsigned 16 bit 2-d vector
- typedef **vector2d**< int32_t > **vector2d_32**

  32 bit 2-d vector
- typedef **vector2d**< uint32_t > **vector2d_u32**

  unsigned 32 bit 2-d vector
- typedef **vector2d**< int64_t > **vector2d_64**

  64 bit 2-d vector
- typedef **vector2d**< uint64_t > **vector2d_u64**

  unsigned 64 bit 2-d vector
- typedef **vector2d**< float > **vector2d_f**

  float 2-d vector
- typedef **vector2d**< double > **vector2d_d**

  double 2-d vector
- typedef **vector3d**< int8_t > **vector3d_8**

  8 bit 3-d vector
- typedef **vector3d**< uint8_t > **vector3d_u8**

  unsigned 8 bit 3-d vector
- typedef **vector3d**< int16_t > **vector3d_16**

  16 bit 3-d vector
- typedef **vector3d**< uint16_t > **vector3d_u16**

  unsigned 16 bit 3-d vector
- typedef **vector3d**< int32_t > **vector3d_32**

  32 bit 3-d vector

- typedef **vector3d**< uint32_t > **vector3d_u32**

  unsigned 32 bit 3-d vector
- typedef **vector3d**< int64_t > **vector3d_64**

  64 bit 3-d vector
- typedef **vector3d**< uint64_t > **vector3d_u64**

  unsigned 64 bit 3-d vector
- typedef **vector3d**< float > **vector3d_f**

  float 3-d vector
- typedef **vector3d**< double > **vector3d_d**

  double 3-d vector
- typedef void(∗ **void_rec**) (void ∗)

  Deletion function typedef.

Enumerations

- enum **setTypes** { **def_set** =0, **small_set**, **sorted_set** }

  Index of abstract data-structures.
- enum **smart_pointer_type** {
  null_type =0, **raw_type**, **shared_type**, **shared_type_array**,
  shared_type_dynamic_delete }

  *Enumeration for types of **os::smart_ptr** (p. 378).*

Functions

- template<class dataType >
  int **defaultCompareSort** (const dataType &v1, const dataType &v2)

  Basic compare.
- template<class dataType >
  int **pointerCompareSort** (**smart_ptr**< dataType > ptr1, **smart_ptr**< dataType > ptr2)

  Raw pointer compare.
- template<class dataType >
  void **quicksort** (dataType ∗arr, unsigned int length, int(∗sort_comparison)(const dataType &, const dataType &)=&**defaultCompareSort**)

  Template quick-sort.
- template<class dataType >
  void **pointerQuicksort** (**smart_ptr**< **smart_ptr**< dataType > > arr, unsigned int length, int(∗sort↩
  _comparison)(**smart_ptr**< dataType >, **smart_ptr**< dataType >)=&**pointerCompareSort**)

  Template for quick-sort, pointer version.
- template<class dataType >
  bool **compareSize** (const **matrix**< dataType > &m1, const **matrix**< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **compareSize** (const **indirectMatrix**< dataType > &m1, const **matrix**< dataType > &m2)

  Compares the size of two matrices.

- template<class dataType >
  bool **compareSize** (const **matrix**< dataType > &m1, const **indirectMatrix**< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **compareSize** (const **indirectMatrix**< dataType > &m1, const **indirectMatrix**< dataType > &m2)

  Compares the size of two matrices.
- template<class dataType >
  bool **testCross** (const **matrix**< dataType > &m1, const **matrix**< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **testCross** (const **indirectMatrix**< dataType > &m1, const **matrix**< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **testCross** (const **matrix**< dataType > &m1, const **indirectMatrix**< dataType > &m2)

  Tests if the cross-product is a legal operation.
- template<class dataType >
  bool **testCross** (const **indirectMatrix**< dataType > &m1, const **indirectMatrix**< dataType > &m2)

  Tests if the cross-product is a legal operation.
- std::ostream & **osout_func** ()

  Standard out object for os namespace.
- std::ostream & **oserr_func** ()

  Standard error object for os namespace.
- template<class targ , class src >
  smart_ptr< targ > **cast** (const **os::smart_ptr**< src > &conv)

  os::smart_ptr *(*p. *378) cast function*

Variables

- std::recursive_mutex * **eventLock**

  Event processing mutex.
- **smart_ptr**< std::ostream > **osout_ptr**

  Standard out pointer for os namespace.
- **smart_ptr**< std::ostream > **oserr_ptr**

  Standard error pointer for os namespace.

11.1.1 Typedef Documentation

typedef **vector2d**<int16_t> **os::vector2d_16**

16 bit 2-d vector

typedef **vector2d**<int32_t> **os::vector2d_32**

32 bit 2-d vector

typedef **vector2d**<int64_t> **os::vector2d_64**

64 bit 2-d vector

typedef **vector2d**<int8_t> **os::vector2d_8**

8 bit 2-d vector

typedef **vector2d**<double> **os::vector2d_d**

double 2-d vector

typedef **vector2d**<float> **os::vector2d_f**

float 2-d vector

typedef **vector2d**<uint16_t> **os::vector2d_u16**

unsigned 16 bit 2-d vector

typedef **vector2d**<uint32_t> **os::vector2d_u32**

unsigned 32 bit 2-d vector

typedef **vector2d**<uint64_t> **os::vector2d_u64**

unsigned 64 bit 2-d vector

typedef **vector2d**<uint8_t> **os::vector2d_u8**

unsigned 8 bit 2-d vector

typedef **vector3d**<int16_t> **os::vector3d_16**

16 bit 3-d vector

typedef **vector3d**<int32_t> **os::vector3d_32**

32 bit 3-d vector

typedef **vector3d**<int64_t> **os::vector3d_64**

64 bit 3-d vector

typedef **vector3d**<int8_t> **os::vector3d_8**

8 bit 3-d vector

typedef **vector3d**<double> **os::vector3d_d**

double 3-d vector

typedef **vector3d**<float> **os::vector3d_f**

float 3-d vector

typedef **vector3d**<uint16_t> **os::vector3d_u16**

unsigned 16 bit 3-d vector

typedef **vector3d**<uint32_t> **os::vector3d_u32**

unsigned 32 bit 3-d vector

typedef **vector3d**<uint64_t> **os::vector3d_u64**

unsigned 64 bit 3-d vector

typedef **vector3d**<uint8_t> **os::vector3d_u8**

unsigned 8 bit 3-d vector

typedef void(∗ os::void_rec) (void ∗)

Deletion function typedef.

The **os::void_rec** (p. 315) function pointer typedef is used by **os::smart_ptr** (p. 378) when it is of type **os::shared_type_dynamic_delete** (p. 316) to destroy non-standard pointers, usually when interfacing with C code.

Parameters

`in,out`	*void∗*	designed for non-standard deletion.

Returns

void

11.1.2 Enumeration Type Documentation

enum **os::setTypes**

Index of abstract data-structures.

This enumeration contains a numbered reference to all of the available abstract data-structures.

Enumerator

def_set Default set enumeration. Currently defaults to a small set.

small_set Small memory burden set. The small set uses an unsorted linked list to store data.

sorted_set Sorted set. The sorted set uses an AVL tree to store data.

enum **os::smart_pointer_type**

Enumeration for types of **os::smart_ptr** (p. 378).

Defines types of **os::smart_ptr** (p. 378). These types are used to define the deletion behaviour of the pointer.

Enumerator

null_type No type. **os::null_type** (p. 316) pointers are the default type of **os::smart_ptr** (p. 378). Any **os::smart_ptr** (p. 378) of type **os::null_type** (p. 316) can be guaranteed to hold a NULL pointer.

raw_type Raw pointer. **os::raw_type** (p. 316) pointers are the default type of **os::smart_ptr** (p. 378) when instantiated with a standard pointer. Any **os::smart_ptr** (p. 378) of type **os::raw_type** (p. 316) is not responsible for the deletion of it's pointer and makes no guarantees as to the availability of it's pointer.

shared_type Reference counted pointer. **os::shared_type** (p. 316) pointers must be instantiated from an **os::smart_ptr** (p. 378) of this type or explicitly through **os::smart_ptr** (p. 378) constructor arguments. **os::shared_type** (p. 316) pointers will automatically delete the pointer contained within the object when the reference count of the **os::smart_ptr** (p. 378) reaches 0.

shared_type_array Reference counted array. Similar in usage and instantiation to **os::raw_type** (p. 316). **os::smart_ptr** (p. 378) of type **os::shared_type_array** (p. 316) are designed to be used with array and will run delete [] when the reference count reaches 0 instead of delete.

shared_type_dynamic_delete Reference pointer with non-standard deletion. Similar in usage and instantiation to **os::raw_type** (p. 316). **os::smart_ptr** (p. 378) of type **os::shared_type_dynamic_delete** (p. 316) are used when the deletion of a pointer is not contained within the object destructor. This is specifically designed for interface with C code not using "new" and "delete."

11.1.3 Function Documentation

template<class targ , class src > **smart_ptr**<targ> os::cast (const **os::smart_ptr**< src > & conv) `[inline]`

os::smart_ptr (p. 378) cast function

Casts an os::smart_ptr<src> to and os::smart_ptr<targ>. This function is a template function, targ and src are the templates respectively. Note that the is an explicit cast and is not guranteed to be safe.

Parameters

`in`	*conv*	Reference to os::smart_ptr<src> to be converted

Returns

New os::smart_ptr<targ> constructed from the received **os::smart_ptr** (p. 378)

template<class dataType > bool os::compareSize (const **matrix**< dataType > & m1, const **matrix**< dataType > & m2)

Compares the size of two matrices.

Compares the size of two raw matrices. If both have the same width and the same height, they are considered to be the same size.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

True if the matrices are the same size

template<class dataType > bool os::compareSize (const **indirectMatrix**< dataType > & m1, const **matrix**< dataType > & m2)

Compares the size of two matrices.

Compares the size of an indirect matrix and a raw matrix in that order. If both have the same width and the same height, they are considered to be the same size.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

True if the matrices are the same size

template<class dataType > bool os::compareSize (const **matrix**< dataType > & m1, const **indirectMatrix**< dataType > & m2)

Compares the size of two matrices.

Compares the size of a raw matrix and an indirect matrix in that order. If both have the same width and the same height, they are considered to be the same size.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

True if the matrices are the same size

template<class dataType > bool os::compareSize (const **indirectMatrix**< dataType > & m1, const **indirectMatrix**< dataType > & m2)

Compares the size of two matrices.

Compares the size of two indirect matrices. If both have the same width and the same height, they are considered to be the same size.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

True if the matrices are the same size

template<class dataType > int os::defaultCompareSort (const dataType & v1, const dataType & v2)

Basic compare.

Acts as a default comparison function for sorting. This function compares the data as if it is in integer form.

Parameters

in	*v1*	Reference 1 to compare
in	*v2*	Reference 2 to compare

Returns

1 if greater than, -1 if less than, 0 if equal to

std::ostream& os::oserr_func ( )

Standard error object for os namespace.

#define statements allow the user to call this function with "os::oserr." Logging is achieved by using "os::oserr" as one would use "std::cerr."

std::ostream& os::osout_func ( )

Standard out object for os namespace.

#define statements allow the user to call this function with "os::osout." Logging is achieved by using "os::osout" as one would use "std::cout."

template<class dataType > int os::pointerCompareSort (**smart_ptr**< dataType > ptr1, **smart_ptr**< dataType > ptr2)

Raw pointer compare.

Acts as a default comparison function for pointer sorting. Compares the raw pointer values of the two arguements and returns the result.

Parameters

in	*ptr1*	Pointer 1 to compare
in	*ptr2*	Pointer 2 to compare

Returns

1 if greater than, -1 if less than, 0 if equal to

template<class dataType > void os::pointerQuicksort (**smart_ptr**< **smart_ptr**< dataType > > arr, unsigned int length, int(∗)(**smart_ptr**< dataType >, **smart_ptr**< dataType >) sort_comparison = &**pointerCompareSort**)

Template for quick-sort, pointer version.

Preforms quick sort on the provided array of the given length where the array is of pointers to the data type instead of the data type.

Parameters

	[in/out]	array Set of data to be sorted
in	*length*	Length of array to be sorted
in	*sort_comparison*	Comparison function definition

Returns

void

template<class dataType > void os::quicksort (dataType ∗ arr, unsigned int length, int(∗)(const dataType &, const dataType &) sort_comparison = &**defaultCompareSort**)

Template quick-sort.

Preforms quick sort on the provided array of the given length with the given comparison function. The default comparison function is one which uses the comparison operators

Parameters

	[in/out]	array Set of data to be sorted
in	*length*	Length of array to be sorted
in	*sort_comparison*	Comparison function definition

Returns

void

template<class dataType > bool os::testCross (const **matrix**< dataType > & m1, const **matrix**< dataType > & m2)

Tests if the cross-product is a legal operation.

Compares the width of the first matrix versus the height of the second. If the two are equal, the cross-product is defined.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Raw matrix reference

Returns

True if the cross-product is defined

template<class dataType > bool os::testCross (const **indirectMatrix**< dataType > & m1, const **matrix**< dataType > & m2)

Tests if the cross-product is a legal operation.

Compares the width of the first matrix versus the height of the second. If the two are equal, the cross-product is defined.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Raw matrix reference

Returns

True if the cross-product is defined

template<class dataType > bool os::testCross (const **matrix**< dataType > & m1, const **indirectMatrix**< dataType > & m2)

Tests if the cross-product is a legal operation.

Compares the width of the first matrix versus the height of the second. If the two are equal, the cross-product is defined.

Parameters

in	*m1*	Raw matrix reference
in	*m2*	Indirect matrix reference

Returns

True if the cross-product is defined

template<class dataType > bool os::testCross (const **indirectMatrix**< dataType > & m1, const **indirectMatrix**< dataType > & m2)

Tests if the cross-product is a legal operation.

Compares the width of the first matrix versus the height of the second. If the two are equal, the cross-product is defined.

Parameters

in	*m1*	Indirect matrix reference
in	*m2*	Indirect matrix reference

Returns

True if the cross-product is defined

11.1.4 Variable Documentation

std::recursive_mutex* os::eventLock

Event processing mutex.

Locks when events are being created, destroyed, bound or triggered. This allows events to be thread safe. The mutex is declared to be recursive to allow for nested event calls.

smart_ptr<std::ostream> os::oserr_ptr

Standard error pointer for os namespace.

This std::ostream is used as standard error for the os namespace. This pointer can be swapped out to programmatically redirect standard error for the os namespace.

smart_ptr<std::ostream> os::osout_ptr

Standard out pointer for os namespace.

This std::ostream is used as standard out for the os namespace. This pointer can be swapped out to programmatically redirect standard out for the os namespace.

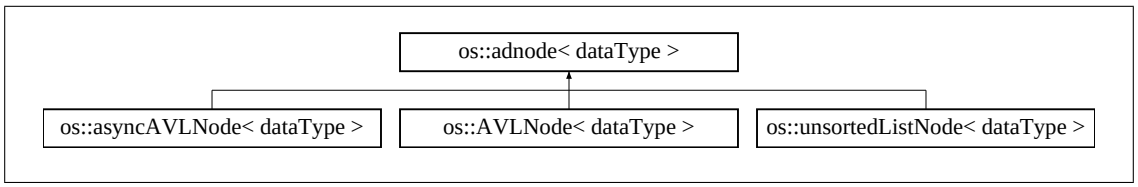
Chapter 12

Class Documentation

12.1 os::adnode< dataType > Class Template Reference

Abstract data-node.

Inheritance diagram for os::adnode< dataType >:



Public Member Functions

- **adnode** (**smart_ptr**< dataType > d)

  Abstract data-node constructor.
- virtual ~**adnode** ()

  Virtual destructor.
- int **compare** (**smart_ptr**< **adnode**< dataType > > inp, bool rawComp=false)

  Compares two abstract data-nodes.
- **smart_ptr**< dataType > & **getData** ()

  Return a reference to the data pointer.
- **smart_ptr**< dataType > & **operator**∗ ()

  Return a reference to the data pointer.
- virtual **smart_ptr**< **adnode**< dataType > > **getNext** ()

  Find the next node.
- virtual **smart_ptr**< **adnode**< dataType > > **getPrev** ()

  Find the previous node.

Protected Attributes

- **smart_ptr**< dataType > **data**

  Data pointer.

12.1.1 Detailed Description

template<class dataType>
class os::adnode< dataType >

Abstract data-node.

A generalized node class used for linked lists, trees, queues and various other abstract data structures. Primarily, this structure is focused on providing access to the node data and allowing traversal of the data-structure.

12.1.2 Constructor & Destructor Documentation

template<class dataType> **os::adnode**< dataType >::**adnode** (**smart_ptr**< dataType > d) `[inline]`

Abstract data-node constructor.

An abstract data-node is meaningless without a pointer to it's dataType. The constructor requires this pointer to initialize the node.

Parameters

`in`	*d*	Data to be bound to the node

template<class dataType> virtual **os::adnode**< dataType >::~**adnode** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.1.3 Member Function Documentation

template<class dataType> int **os::adnode**< dataType >::compare (**smart_ptr**< **adnode**< dataType > > inp, bool rawComp = `false`) `[inline]`

Compares two abstract data-nodes.

Abstract data nodes use the comparison functions defined by their data pointers to determine their comparison

Parameters

`in`	*inp*	Data-node being compared with

Returns

1, 0, -1 (Greater than, equal to, less than)

template<class dataType> **smart_ptr**<dataType>& **os::adnode**< dataType >::getData ( ) `[inline]`

Return a reference to the data pointer.

Returns

adnode<**datqType**>**::data** (p. 325)

template<class dataType> virtual **smart_ptr**<**adnode**<dataType> > **os::adnode**< dataType >::getNext ( ) `[inline]`, `[virtual]`

Find the next node.

This functions attempts to search for the next node in the structure. By default, or if this node either cannot be found or does not exist, a NULL pointer is returned.

Returns

Pointer to the next node in the structure

Reimplemented in **os::asyncAVLNode**< **dataType** > (p. 332), **os::asyncAVLNode**< **sender**↩**Type** > (p. 332), **os::asyncAVLNode**< **receiverType** > (p. 332), **os::AVLNode**< **dataType** > (p. 345), and **os::unsortedListNode**< **dataType** > (p. 398).

template<class dataType> virtual **smart_ptr**<**adnode**<dataType> > **os::adnode**< dataType >::getPrev ( ) `[inline]`, `[virtual]`

Find the previous node.

This functions attempts to search for the previous node in the structure. By default, or if this node either cannot be found or does not exist, a NULL pointer is returned.

Returns

Pointer to the previous node in the structure

Reimplemented in **os::AVLNode**< **dataType** > (p. 346), **os::asyncAVLNode**< **dataType** > (p. 332), **os::asyncAVLNode**< **senderType** > (p. 332), **os::asyncAVLNode**< **receiverType** > (p. 332), and **os::unsortedListNode**< **dataType** > (p. 399).

template<class dataType> **smart_ptr**<dataType>& **os::adnode**< dataType >::operator∗ ( ) `[inline]`

Return a reference to the data pointer.

Returns

adnode<**datqType**>**::data** (p. 325)

12.1.4 Member Data Documentation

template<class dataType> **smart_ptr**<dataType> **os::adnode**< dataType >::data `[protected]`

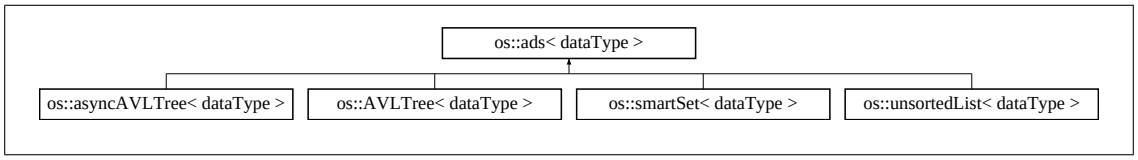
Data pointer.

A pointer to the data being held by the node. This is used to compare nodes as well.

12.2 os::ads< dataType > Class Template Reference

Abstract datastructure.

Inheritance diagram for os::ads< dataType >:



Public Member Functions

- **ads** ()

  Default constructor.
- virtual ~**ads** ()

  Virtual destructor.
- virtual bool **insert** (**smart_ptr**< dataType > x)

  Inserts a data pointer.
- virtual unsigned int **size** () const

  Returns the number of elements in the datastructure.
- virtual **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< dataType > x)

  Finds a matching node.
- virtual bool **findDelete** (**smart_ptr**< dataType > x)

  Finds a matching node and removes it.
- virtual **smart_ptr**< **adnode**< dataType > > **getFirst** ()

  Returns the first node.
- virtual **smart_ptr**< **adnode**< dataType > > **getLast** ()

  Returns the last node.
- virtual bool **insert** (**smart_ptr**< **ads**< dataType > > x)

  Inserts an entire datastructure.
- bool **rawInsert** (**smart_ptr**< dataType > x)

  Inserts a data pointer.
- bool **rawCompare** () const

  Return state of raw compare.
- void **setRawCompare** (bool rwcmp)

  Set raw-compare.

Protected Attributes

- bool **_rawCompare**

  Allows for raw compare data-structures.

12.2.1 Detailed Description

template<class dataType>
class os::ads< dataType >

Abstract datastructure.

A generalized datastructure class which acts as an interface for all datastructures classes. If not extended, the abstract datastructures class is useless.

12.2.2 Constructor & Destructor Documentation

template<class dataType> **os::ads**< dataType >::**ads** ( ) `[inline]`

Default constructor.

This constructor does nothing, as there are no objects to initialize.

template<class dataType> virtual **os::ads**< dataType >::~**ads** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.2.3 Member Function Documentation

template<class dataType> virtual **smart_ptr**<**adnode**<dataType> > **os::ads**< dataType >::find (**smart_ptr**< dataType > x) `[inline]`, `[virtual]`

Finds a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). Each datastructure which inherits from this class will re-implement this function.

[in] x dataType pointer to be compared against

Returns

The found node if applicable, else NULL

Reimplemented in **os::AVLTree**< **dataType** > (p. 352), **os::asyncAVLTree**< **dataType** > (p. 339), **os::asyncAVLTree**< **senderType** > (p. 339), **os::asyncAVLTree**< **receiverType** > (p. 339), **os::unsortedList**< **dataType** > (p. 395), and **os::smartSet**< **dataType** > (p. 391).

template<class dataType> virtual bool **os::ads**< dataType >::findDelete (**smart_ptr**< dataType > x) [inline], [virtual]

Finds a matching node and removes it.

Finds a pointer to an object of type "dataType" given a comparison pointer. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). Each datastructure which inherits from this class will re-implement this function. After finding a node, it will be removed from the datastructure.

[in] x dataType pointer to be compared against

Returns

true if the node was found and deleted, else false

Reimplemented in **os::AVLTree**< **dataType** > (p. 353), **os::asyncAVLTree**< **dataType** > (p. 340), **os::asyncAVLTree**< **senderType** > (p. 340), **os::asyncAVLTree**< **receiverType** > (p. 340), **os::unsortedList**< **dataType** > (p. 395), and **os::smartSet**< **dataType** > (p. 391).

template<class dataType> virtual **smart_ptr**<**adnode**<dataType> > **os::ads**< dataType >::getFirst ( ) [inline], [virtual]

Returns the first node.

Each datastructure has a different definition of what defines "first." By default, this function returns NULL. Datastructures which inherit from this class must re-implement this function.

Returns

The first node, if it exists

Reimplemented in **os::asyncAVLTree**< **dataType** > (p. 341), **os::asyncAVLTree**< **senderType** > (p. 341), **os::asyncAVLTree**< **receiverType** > (p. 341), **os::AVLTree**< **dataType** > (p. 354), **os::unsortedList**< **dataType** > (p. 395), and **os::smartSet**< **dataType** > (p. 391).

template<class dataType> virtual **smart_ptr**<**adnode**<dataType> > **os::ads**< dataType >::getLast ( ) [inline], [virtual]

Returns the last node.

Each datastructure has a different definition of what defines "last." By default, this function returns NULL. Datastructures which inherit from this class must re-implement this function.

Returns

The last node, if it exists

Reimplemented in **os::asyncAVLTree**< **dataType** > (p. 341), **os::asyncAVLTree**< **senderType** > (p. 341), **os::asyncAVLTree**< **receiverType** > (p. 341), **os::AVLTree**< **dataType** > (p. 354), **os::unsortedList**< **dataType** > (p. 396), and **os::smartSet**< **dataType** > (p. 392).

template<class dataType> virtual bool **os::ads**< dataType >::insert (**smart_ptr**< dataType > x) [inline], [virtual]

Inserts a data pointer.

Inserts a pointer to an object of type "dataType." Each datastructure which inherits from this class will re-implement this function

[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

Reimplemented in **os::AVLTree**< **dataType** > (p. 355), **os::asyncAVLTree**< **dataType** > (p. 342), **os::asyncAVLTree**< **senderType** > (p. 342), **os::asyncAVLTree**< **receiverType** > (p. 342), **os::unsortedList**< **dataType** > (p. 396), and **os::smartSet**< **dataType** > (p. 392).

template<class dataType> virtual bool **os::ads**< dataType >::insert (**smart_ptr**< **ads**< dataType > > x) [inline], [virtual]

Inserts an entire datastructure.

This function may be redefined to speed-up insertion. Currently, this function will be O(n * insertionTime) where n is the number of elements in x

[in] x datastructure of type dataType to be inserted

Returns

true if successful, false if failed

Reimplemented in **os::AVLTree**< **dataType** > (p. 355), **os::asyncAVLTree**< **dataType** > (p. 341), **os::asyncAVLTree**< **senderType** > (p. 341), **os::asyncAVLTree**< **receiverType** > (p. 341), **os::unsortedList**< **dataType** > (p. 396), and **os::smartSet**< **dataType** > (p. 392).

template<class dataType> bool **os::ads**< dataType >::rawCompare ( ) const [inline]

Return state of raw compare.

Returns

_rawCompare

template<class dataType> bool **os::ads**< dataType >::rawInsert (**smart_ptr**< dataType > x) [inline]

Inserts a data pointer.

Inserts a pointer to an object of type "dataType." This function disabiguates certain calls to insert.

[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

template<class dataType> void **os::ads**< dataType >::setRawCompare (bool rwcmp) [inline]

Set raw-compare.

Parameters

in	*rwcmp*	Value of raw compare to set

Returns

void

template<class dataType> virtual unsigned int **os::ads**< dataType >::size ( ) const [inline], [virtual]

Returns the number of elements in the datastructure.

This function must be re-implemented by all classes which inherit from this class. By default, this function returns 0.

Returns

number of elements as an unsigned integer

Reimplemented in **os::asyncAVLTree**< **dataType** > (p. 342), **os::asyncAVLTree**< **senderType** > (p. 342), **os::asyncAVLTree**< **receiverType** > (p. 342), **os::AVLTree**< **dataType** > (p. 355), **os::unsortedList**< **dataType** > (p. 396), and **os::smartSet**< **dataType** > (p. 393).

12.2.4 Member Data Documentation

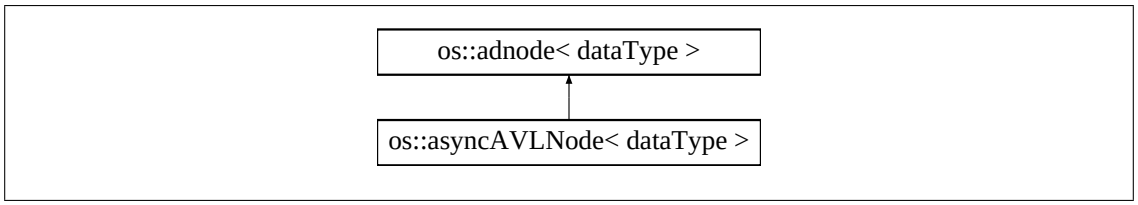
template<class dataType> bool **os::ads**< dataType >::_rawCompare [protected]

Allows for raw compare data-structures.

12.3 os::asyncAVLNode< dataType > Class Template Reference

Node for usage in an asynchronous AVL tree.

Inheritance diagram for os::asyncAVLNode< dataType >:



Public Member Functions

- **asyncAVLNode** (**smart_ptr**< dataType > d, **asyncAVLTree**< dataType > *master)

  Abstract data-node constructor.
- virtual ~**asyncAVLNode** ()

  Virtual destructor.
- **smart_ptr**< **adnode**< dataType > > **getNext** ()

  Find the next node.
- **smart_ptr**< **adnode**< dataType > > **getPrev** ()

  Find the previous node.

Protected Member Functions

- **smart_ptr**< **asyncAVLNode**< dataType > > **getParent** ()

  Returns the parent node.
- **smart_ptr**< **asyncAVLNode**< dataType > > **getChild** (int x)

  Returns a child by index.
- int **getHeight** () const

  Returns the height of the sub-tree.
- void **setHeight** ()

  Sets the height of the sub-tree.
- void **setChild** (**smart_ptr**< **asyncAVLNode**< dataType > > c, bool _rawCompare)

  Add a child to this node.
- void **setParent** (**smart_ptr**< **asyncAVLNode**< dataType > > p, **smart_ptr**< **asyncAVLNode**< dataType > > self_pointer, bool _rawCompare)

  Sets the parent node.
- void **removeChild** (**smart_ptr**< **asyncAVLNode**< dataType > > c, bool _rawCompare)

  Remove a child from this node.
- void **removeChild** (int pos)

  Remove a child from this node.
- void **removeParent** ()

  Remove the parent node.
- void **remove** ()

  Remove all children and parents.

Protected Attributes

- **smart_ptr**< **asyncAVLNode**< dataType > > **parent**

  Parent node one level up in the tree.
- **smart_ptr**< **asyncAVLNode**< dataType > > **child1**

  Left child one level down in the tree.
- **smart_ptr**< **asyncAVLNode**< dataType > > **child2**

  Right child one level down in the tree.
- int **height**

  The height of the tree.
- **asyncAVLTree**< dataType > ∗ **masterTree**

  Reference to source tree.

Friends

- class **asyncAVLTree**< **dataType** >

  AVL Tree must know details of node implementation.

12.3.1 Detailed Description

template<class dataType>
class os::asyncAVLNode< dataType >

Node for usage in an asynchronous AVL tree.

The AVL node class implements a number of functions unique to an AVL tree. This node has knowledge of the structure of the AVL tree through its parent and children.

12.3.2 Constructor & Destructor Documentation

template<class dataType> **os::asyncAVLNode**< dataType >::**asyncAVLNode** (**smart_ptr**< dataType > d, **asyncAVLTree**< dataType > * master) [inline]

Abstract data-node constructor.

An AVL node is meaningless without a pointer to it's dataType. The constructor requires this pointer to initialize the node. Parent and children nodes are, by default, initialized to 0.

Parameters

in	*d*	Data to be bound to the node

template<class dataType> virtual **os::asyncAVLNode**< dataType >::~**asyncAVLNode** ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.3.3 Member Function Documentation

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLNode**< dataType >::getChild (int x) [inline], [protected]

Returns a child by index.

Returns child node by index. 0 indicates the left child, **asyncAVLNode**<**dataType**>**::child1** (p. 334). 1 indicates the right child, **asyncAVLNode**<**dataType**>**::child2** (p. 334). All other indices will return NULL.

Returns

os::asyncAVLNode<**dataType**>**::child1** (p. 334) for x==0, **asyncAVLNode**<**dataType**>**::child2** (p. 334) for x==1

template<class dataType> int **os::asyncAVLNode**< dataType >::getHeight ( ) const [inline], [protected]

Returns the height of the sub-tree.

Returns

os::asyncAVLNode<**dataType**>**::height** (p. 335)

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLNode**< dataType >::getNext ( ) [virtual]

Find the next node.

This functions attempts to search for the next node in the structure. This trips the traverse flag of the current node and traverses the tree looking for the next node.

Returns

Pointer to the next node in the structure

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLNode**< dataType >::getParent ( ) [inline], [protected]

Returns the parent node.

Returns

os::asyncAVLNode<**dataType**>**::parent** (p. 335)

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLNode**< dataType >::getPrev ( ) [virtual]

Find the previous node.

This functions attempts to search for the previous node in the structure. This trips the traverse flag of the current node and traverses the tree looking for the previous node.

Returns

Pointer to the previous node in the structure

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType> void **os::asyncAVLNode**< dataType >::remove ( ) [inline], [protected]

Remove all children and parents.

This function is important because nodes are of type **os::smart_ptr** (p. 378), since there are co-dependencies, failure to run this function on deletion of the tree will cause a memory leak.

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::removeChild (**smart_ptr**< **asyncAVLNode**< dataType > > c, bool _rawCompare) [inline], [protected]

Remove a child from this node.

Checks **os::asyncAVLNode**<**dataType**>**::child1** (p. 334) and **os::asyncAVLNode**<**dataType**>↩ **::child2** (p. 334) for equality with the the node received as a parameter.

Parameters

in	*c*	Node to be removed

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::removeChild (int pos) [inline], [protected]

Remove a child from this node.

Remove **os::asyncAVLNode**<**dataType**>**::child1** (p. 334) if position is 0 and **os::asyncAVL↩Node**<**dataType**>**::child2** (p. 334) if position is 1.

Parameters

in	*pos*	Node index to be removed

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::removeParent ( ) [inline], [protected]

Remove the parent node.

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::setChild (**smart_ptr**< **asyncAVLNode**< dataType > > c, bool _rawCompare) [inline], [protected]

Add a child to this node.

Set **os::asyncAVLNode**<**dataType**>**::child1** (p. 334) or **os::asyncAVLNode**<**dataType**>**::child2** (p. 334) based on the comparison of the node to be inserted with the current node.

Parameters

in	*c*	Node to be inserted

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::setHeight ( ) [inline], [protected]

Sets the height of the sub-tree.

Uses the height of the sub-tree of the node's children to calculate the heigh of the sub-tree of this node.

Returns

void

template<class dataType> void **os::asyncAVLNode**< dataType >::setParent (**smart_ptr**< **asyncAVLNode**< dataType > > p, **smart_ptr**< **asyncAVLNode**< dataType > > self_pointer, bool _rawCompare) [inline], [protected]

Sets the parent node.

Sets the parent node of the current node. This function requires a pointer to the current node for memory management.

Parameters

in	*p*	Parent node
in	*self_pointer*	Pointer to self, with memory management

Returns

void

12.3.4 Friends And Related Function Documentation

template<class dataType> friend class **asyncAVLTree**< dataType > [friend]

AVL Tree must know details of node implementation.

Since the AVL node implements many of the unique functions of the AVL tree, the tree must be aware of the private members of it's nodes.

12.3.5 Member Data Documentation

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLNode**< dataType >::child1 [protected]

Left child one level down in the tree.

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLNode**< dataType >::child2 [protected]

Right child one level down in the tree.

template<class dataType> int **os::asyncAVLNode**< dataType >::height [protected]

The height of the tree.

This variable is kept to reduce computation time. It is dependent on the height of a node's children nodes. The **asyncAVLNode**<**dataType**>**::setHeight()** (p. 334) resets the height based on the height of the node's children.

template<class dataType> **asyncAVLTree**<dataType>* **os::asyncAVLNode**< dataType >::masterTree [protected]

Reference to source tree.

This reference to the source tree is used when incrementing or decrementing the node, locking the tree temporarily.

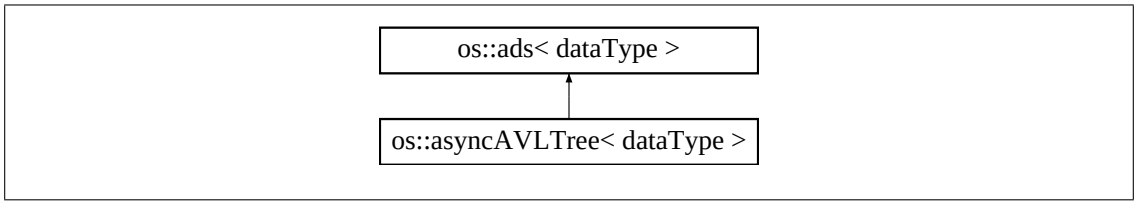
template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLNode**< dataType >::parent [protected]

Parent node one level up in the tree.

12.4 os::asyncAVLTree< dataType > Class Template Reference

Asynchronous balanced binary search tree.

Inheritance diagram for os::asyncAVLTree< dataType >:



Public Member Functions

- **asyncAVLTree** ()

  Default constructor.
- virtual ~**asyncAVLTree** ()

  Virtual destructor.
- bool **insert** (**smart_ptr**< **ads**< dataType > > x)

  Inserts an os::ads<dataType>
- bool **insert** (**smart_ptr**< dataType > x)

  Inserts a data node.
- **smart_ptr**< **asyncAVLNode**< dataType > > **getRoot** ()

  Return the root of the tree.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< dataType > x)

  Finds a matching node.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< **adnode**< dataType > > x)

Finds by adnode node.

- **smart_ptr**< **asyncAVLNode**< dataType > > **find** (**smart_ptr**< **asyncAVLNode**< dataType > > x)

  *Finds by **asyncAVLNode** (p. 329) node.*
- bool **findDelete** (**smart_ptr**< dataType > x)

  Finds and delete a matching node.
- bool **findDelete** (long x)

  Finds and delete a matching node.
- bool **findDelete** (**smart_ptr**< **asyncAVLNode**< dataType > > x)

  Finds and delete by node.
- virtual unsigned int **size** () const

  Finds and delete a matching node.
- **smart_ptr**< **adnode**< dataType > > **getFirst** ()

  Returns the first node.
- **smart_ptr**< **adnode**< dataType > > **getLast** ()

  Returns the last node.

Protected Member Functions

- bool **balanceDelete** (**smart_ptr**< **asyncAVLNode**< dataType > > x, bool **_rawCompare**)

  Removes a node and balances the tree.
- bool **checkBalance** (**smart_ptr**< **asyncAVLNode**< dataType > > x)

  Checks if a sub-tree is balanced.
- void **balanceUp** (**smart_ptr**< **asyncAVLNode**< dataType > > x)

  Balances this node and ancestor nodes.
- bool **balance** (**smart_ptr**< **asyncAVLNode**< dataType > > x)

  Balances a single node.
- bool **singleRotation** (**smart_ptr**< **asyncAVLNode**< dataType > > r, int dir)

  Rotates a node.
- bool **doubleRotation** (**smart_ptr**< **asyncAVLNode**< dataType > > r, int dir)

  Double-rotate a node.
- **smart_ptr**< **asyncAVLNode**< dataType > > **findBottom** (**smart_ptr**< **asyncAVLNode**< data↩ Type > > x, int dir)

  Find first or last node in a tree.

Protected Attributes

- **smart_ptr**< **asyncAVLNode**< dataType > > **root**

  Root node of the tree.
- unsigned int **numElements**

  Number of elements in the tree.
- std::mutex **mtx**

  Mutex to ensure synchronous access.

Friends

- class **asyncAVLNode**< **dataType** >

  AVL Node must have access to mutex.

12.4.1 Detailed Description

template<class dataType>
class os::asyncAVLTree< dataType >

Asynchronous balanced binary search tree.

The AVL Tree rigorously balances a binary search tree. As a template class, it can hold any kind of dataType so long as the data type implements basic comparison functions.

12.4.2 Constructor & Destructor Documentation

template<class dataType> **os::asyncAVLTree**< dataType >::**asyncAVLTree** ( ) `[inline]`

Default constructor.

Sets the number of elements to 0 and the root to NULL.

template<class dataType> virtual **os::asyncAVLTree**< dataType >::~**asyncAVLTree** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. The AVL tree must explicitly force deletion through the **asyncAVLNode**<**dataType**>**::remove()** (p. 332) function.

12.4.3 Member Function Documentation

template<class dataType> bool **os::asyncAVLTree**< dataType >::balance (**smart_ptr**< **asyncAVLNode**< dataType > > x) `[inline]`, `[protected]`

Balances a single node.

Parameters

`in`	*x*	Node to be balanced

Returns

true if the node is already balanced, else, false

template<class dataType> bool **os::asyncAVLTree**< dataType >::balanceDelete (**smart_ptr**< **asyncAVLNode**< dataType > > x, bool _rawCompare) `[inline]`, `[protected]`

Removes a node and balances the tree.

Must receive as an argument a node in the tree. This function removes the node from the tree and re-balances the tree.

Parameters

in	*x*	Node to be deleted

Returns

true if successful, false if failed

template<class dataType> void **os::asyncAVLTree**< dataType >::balanceUp (**smart_ptr**< **asyncAVLNode**< dataType > > x) [inline], [protected]

Balances this node and ancestor nodes.

Balances the current node then orders it's parent node to be balanced as well. This process continues until a node has no parent (indicating the node is the root)

Parameters

in	*x*	Node to be balanced

Returns

void

template<class dataType> bool **os::asyncAVLTree**< dataType >::checkBalance (**smart_ptr**< **asyncAVLNode**< dataType > > x) [inline], [protected]

Checks if a sub-tree is balanced.

Checks if the received node is balanced. This operation is inexpensive as it merely involves comparing the heights of the children nodes.

Parameters

in	*x*	Node to be checked

Returns

true if balanced, false if not

template<class dataType> bool **os::asyncAVLTree**< dataType >::doubleRotation (**smart_ptr**< **asyncAVLNode**< dataType > > r, int dir) [inline], [protected]

Double-rotate a node.

Double-rotates a node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Node to be rotated
in	*dir*	Direction node is to be rotated

Returns

true if successful, else, false

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLTree**< dataType >::find (**smart_ptr**< dataType > x) [inline], [virtual]

Finds a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 326).

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLTree**< dataType >::find (**smart_ptr**< **adnode**< dataType > > x) [inline]

Finds by adnode node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::adnode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLTree**< dataType >::find (**smart_ptr**< **asyncAVLNode**< dataType > > x) [inline]

Finds by **asyncAVLNode** (p. 329) node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::asyncAVLNode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLTree**< dataType >::findBottom (**smart_ptr**< **asyncAVLNode**< dataType > > x, int dir) [inline], [protected]

Find first or last node in a tree.

Finds the first or last node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Starting node
in	*dir*	Direction node to search in

Returns

First or last node in sub-tree

template<class dataType> bool **os::asyncAVLTree**< dataType >::findDelete (**smart_ptr**< dataType > x) [inline], [virtual]

Finds and delete a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x dataType pointer to be compared against

Returns

true if the node was found and deleted, else false

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType> bool **os::asyncAVLTree**< dataType >::findDelete (long x) [inline]

Finds and delete a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x dataType pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType> bool **os::asyncAVLTree**< dataType >::findDelete (**smart_ptr**< **asyncAVLNode**< dataType > > x) [inline]

Finds and delete by node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<data↩ Type> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::asyncAVLNode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLTree**< dataType >::getFirst ( ) [inline], [virtual]

Returns the first node.

For the AVL tree, the first node is defined as the child at index 1. Note that while an os::adnode<dataType> is returned, the true type of the pointer returned is os::asyncAVLNode<dataType>. This function is O(log(n)).

Returns

The first node, if it exists

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType> **smart_ptr**<**adnode**<dataType> > **os::asyncAVLTree**< dataType >::getLast ( ) [inline], [virtual]

Returns the last node.

For the AVL tree, the last node is defined as the child at index 0. Note that while an os::adnode<dataType> is returned, the true type of the pointer returned is os::asyncAVLNode<dataType>. This function is O(log(n)).

Returns

The last node, if it exists

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLTree**< dataType >::getRoot ( ) [inline]

Return the root of the tree.

Returns

os::asyncAVLTree<**dataType**>**::root** (p. 343)

template<class dataType> bool **os::asyncAVLTree**< dataType >::insert (**smart_ptr**< **ads**< dataType > > x) [inline], [virtual]

Inserts an os::ads<dataType>

Inserts every element in a given abstract datastructure into this tree. Adopts the insertion function of os::ads<dataType>

[in] x pointer to os::ads<dataType>

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 328).

template<class dataType> bool **os::asyncAVLTree**< dataType >::insert (**smart_ptr**< dataType > x) [inline], [virtual]

Inserts a data node.

Inserts a pointer to an object of type "dataType." This insertion will place the node into the binary tree and balance the tree. This function takes O(log(n)) where n is the number of elements in the tree.

[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType> bool **os::asyncAVLTree**< dataType >::singleRotation (**smart_ptr**< **asyncAVLNode**< dataType > > r, int dir) [inline], [protected]

Rotates a node.

Rotates a node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Node to be rotated
in	*dir*	Direction node is to be rotated

Returns

true if successful, else, false

template<class dataType> virtual unsigned int **os::asyncAVLTree**< dataType >::size ( ) const [inline], [virtual]

Finds and delete a matching node.

Returns

os::asyncAVLTree<**dataType**>**::numElements** (p. 343)

Reimplemented from **os::ads**< **dataType** > (p. 329).

12.4.4 Friends And Related Function Documentation

template<class dataType> friend class **asyncAVLNode**< dataType > [friend]

AVL Node must have access to mutex.

When the **AVLNode** (p. 343) finds the next element or finds the previous element, it must lock the mutex to prevent insertion and deletion into the tree.

12.4.5 Member Data Documentation

template<class dataType> std::mutex **os::asyncAVLTree**< dataType >::mtx `[protected]`

Mutex to ensure synchronous access.

template<class dataType> unsigned int **os::asyncAVLTree**< dataType >::numElements `[protected]`

Number of elements in the tree.

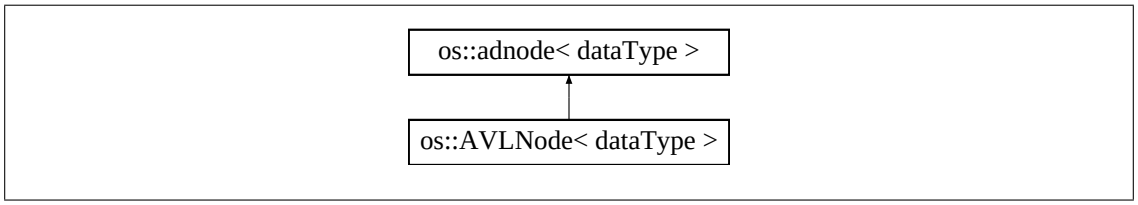
template<class dataType> **smart_ptr**<**asyncAVLNode**<dataType> > **os::asyncAVLTree**< dataType >::root `[protected]`

Root node of the tree.

12.5 os::AVLNode< dataType > Class Template Reference

Node for usage in an AVL tree.

Inheritance diagram for os::AVLNode< dataType >:



Public Member Functions

- **AVLNode** (**smart_ptr**< dataType > d)
  Abstract data-node constructor.
- virtual ~**AVLNode** ()
  Virtual destructor.
- **smart_ptr**< **adnode**< dataType > > **getNext** ()
  Find the next node.
- **smart_ptr**< **adnode**< dataType > > **getPrev** ()
  Find the previous node.

Protected Member Functions

- **smart_ptr**< **AVLNode**< dataType > > **getParent** ()
  Returns the parent node.
- **smart_ptr**< **AVLNode**< dataType > > **getChild** (int x)
  Returns a child by index.
- int **getHeight** () const

Returns the height of the sub-tree.

- void **setHeight** ()

  Sets the height of the sub-tree.
- void **setChild** (**smart_ptr**< **AVLNode**< dataType > > c)

  Add a child to this node.
- void **setParent** (**smart_ptr**< **AVLNode**< dataType > > p, **smart_ptr**< **AVLNode**< dataType > > self_pointer)

  Sets the parent node.
- void **removeChild** (**smart_ptr**< **AVLNode**< dataType > > c)

  Remove a child from this node.
- void **removeChild** (int pos)

  Remove a child from this node.
- void **removeParent** ()

  Remove the parent node.
- void **remove** ()

  Remove all children and parents.

Protected Attributes

- **smart_ptr**< **AVLNode**< dataType > > **parent**

  Parent node one level up in the tree.
- **smart_ptr**< **AVLNode**< dataType > > **child1**

  Left child one level down in the tree.
- **smart_ptr**< **AVLNode**< dataType > > **child2**

  Right child one level down in the tree.
- int **height**

  The height of the tree.

Friends

- class **AVLTree**< **dataType** >

  AVL Tree must know details of node implementation.

12.5.1 Detailed Description

template<class dataType>
class os::AVLNode< dataType >

Node for usage in an AVL tree.

The AVL node class implements a number of functions unique to an AVL tree. This node has knowledge of the structure of the AVL tree through its parent and children.

12.5.2 Constructor & Destructor Documentation

template<class dataType > **os::AVLNode**< dataType >::**AVLNode** (**smart_ptr**< dataType > d) `[inline]`

Abstract data-node constructor.

An AVL node is meaningless without a pointer to it's dataType. The constructor requires this pointer to initialize the node. Parent and children nodes are, by default, initialized to 0.

Parameters

`in`	*d*	Data to be bound to the node

template<class dataType > virtual **os::AVLNode**< dataType >::~**AVLNode** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.5.3 Member Function Documentation

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLNode**< dataType >::getChild (int x) `[inline]`, `[protected]`

Returns a child by index.

Returns child node by index. 0 indicates the left child, **AVLNode**<**dataType**>**::child1** (p. 348). 1 indicates the right child, **AVLNode**<**dataType**>**::child2** (p. 348). All other indices will return NULL.

Returns

os::AVLNode<**dataType**>**::child1** (p. 348) for x==0, **AVLNode**<**dataType**>**::child2** (p. 348) for x==1

template<class dataType > int **os::AVLNode**< dataType >::getHeight ( ) const `[inline]`, `[protected]`

Returns the height of the sub-tree.

Returns

os::AVLNode<**dataType**>**::height** (p. 348)

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLNode**< dataType >::getNext ( ) `[inline]`, `[virtual]`

Find the next node.

This functions attempts to search for the next node in the structure. This trips the traverse flag of the current node and traverses the tree looking for the next node.

Returns

Pointer to the next node in the structure

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLNode**< dataType >::getParent ( ) [inline], [protected]

Returns the parent node.

Returns

os::AVLNode<**dataType**>**::parent** (p. 348)

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLNode**< dataType >::getPrev ( ) [inline], [virtual]

Find the previous node.

This functions attempts to search for the previous node in the structure. This trips the traverse flag of the current node and traverses the tree looking for the previous node.

Returns

Pointer to the previous node in the structure

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType > void **os::AVLNode**< dataType >::remove ( ) [inline], [protected]

Remove all children and parents.

This function is important because nodes are of type **os::smart_ptr** (p. 378), since there are co-dependencies, failure to run this function on deletion of the tree will cause a memory leak.

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::removeChild (**smart_ptr**< **AVLNode**< dataType > > c) [inline], [protected]

Remove a child from this node.

Checks **os::AVLNode**<**dataType**>**::child1** (p. 348) and **os::AVLNode**<**dataType**>**::child2** (p. 348) for equality with the the node received as a parameter.

Parameters

in	*c*	Node to be removed

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::removeChild (int pos) [inline], [protected]

Remove a child from this node.

Remove **os::AVLNode**<**dataType**>**::child1** (p. 348) if position is 0 and **os::AVLNode**<**dataType**>**::child2** (p. 348) if position is 1.

Parameters

in	*pos*	Node index to be removed

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::removeParent ( ) [inline], [protected]

Remove the parent node.

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::setChild (**smart_ptr**< **AVLNode**< dataType > > c) [inline], [protected]

Add a child to this node.

Set **os::AVLNode**<**dataType**>**::child1** (p. 348) or **os::AVLNode**<**dataType**>**::child2** (p. 348) based on the comparison of the node to be inserted with the current node.

Parameters

in	*c*	Node to be inserted

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::setHeight ( ) [inline], [protected]

Sets the height of the sub-tree.

Uses the height of the sub-tree of the node's children to calculate the heigh of the sub-tree of this node.

Returns

void

template<class dataType > void **os::AVLNode**< dataType >::setParent (**smart_ptr**< **AVLNode**< dataType > > p, **smart_ptr**< **AVLNode**< dataType > > self_pointer) [inline], [protected]

Sets the parent node.

Sets the parent node of the current node. This function requires a pointer to the current node for memory management.

Parameters

in	*p*	Parent node
in	*self_pointer*	Pointer to self, with memory management

Returns

void

12.5.4 Friends And Related Function Documentation

template<class dataType > friend class **AVLTree**< dataType > [friend]

AVL Tree must know details of node implementation.

Since the AVL node implements many of the unique functions of the AVL tree, the tree must be aware of the private members of it's nodes.

12.5.5 Member Data Documentation

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLNode**< dataType >::child1 [protected]

Left child one level down in the tree.

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLNode**< dataType >::child2 [protected]

Right child one level down in the tree.

template<class dataType > int **os::AVLNode**< dataType >::height [protected]

The height of the tree.

This variable is kept to reduce computation time. It is dependent on the height of a node's children nodes. The **AVLNode**<**dataType**>**::setHeight()** (p. 347) resets the height based on the height of the node's children.

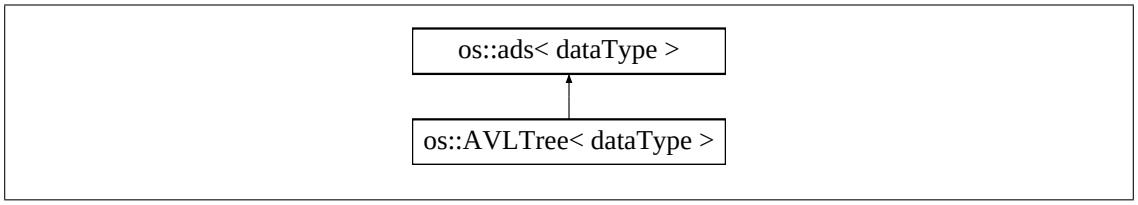
template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLNode**< dataType >::parent [protected]

Parent node one level up in the tree.

12.6 os::AVLTree< dataType > Class Template Reference

Balanced binary search tree.

Inheritance diagram for os::AVLTree< dataType >:



Public Member Functions

- **AVLTree** ()

  Default constructor.
- virtual ~**AVLTree** ()

  Virtual destructor.
- bool **insert** (**smart_ptr**< **ads**< dataType > > x)

  Inserts an os::ads<dataType>
- bool **insert** (**smart_ptr**< dataType > x)

  Inserts a data node.
- **smart_ptr**< **AVLNode**< dataType > > **getRoot** ()

  Return the root of the tree.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< dataType > x)

  Finds a matching node.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< **adnode**< dataType > > x)

  Finds by adnode node.
- **smart_ptr**< **AVLNode**< dataType > > **find** (**smart_ptr**< **AVLNode**< dataType > > x)

  *Finds by **AVLNode** (p. 343) node.*
- bool **findDelete** (**smart_ptr**< dataType > x)

  Finds and delete a matching node.
- bool **findDelete** (**smart_ptr**< **AVLNode**< dataType > > x)

  Finds and delete by node.
- virtual unsigned int **size** () const

  Finds and delete a matching node.
- **smart_ptr**< **adnode**< dataType > > **getFirst** ()

  Returns the first node.
- **smart_ptr**< **adnode**< dataType > > **getLast** ()

  Returns the last node.

Protected Member Functions

- bool **balanceDelete** (**smart_ptr**< **AVLNode**< dataType > > x)

  Removes a node and balances the tree.
- bool **checkBalance** (**smart_ptr**< **AVLNode**< dataType > > x)

  Checks if a sub-tree is balanced.
- void **balanceUp** (**smart_ptr**< **AVLNode**< dataType > > x)

  Balances this node and ancestor nodes.
- bool **balance** (**smart_ptr**< **AVLNode**< dataType > > x)

  Balances a single node.
- bool **singleRotation** (**smart_ptr**< **AVLNode**< dataType > > r, int dir)

  Rotates a node.
- bool **doubleRotation** (**smart_ptr**< **AVLNode**< dataType > > r, int dir)

  Double-rotate a node.
- **smart_ptr**< **AVLNode**< dataType > > **findBottom** (**smart_ptr**< **AVLNode**< dataType > > x, int dir)

  Find first or last node in a tree.

Protected Attributes

- **smart_ptr**< **AVLNode**< dataType > > **root**

  Root node of the tree.
- unsigned int **numElements**

  Number of elements in the tree.

12.6.1 Detailed Description

template<class dataType>
class os::AVLTree< dataType >

Balanced binary search tree.

The AVL Tree rigorously balances a binary search tree. As a template class, it can hold any kind of dataType so long as the data type implements basic comparison functions.

12.6.2 Constructor & Destructor Documentation

template<class dataType > **os::AVLTree**< dataType >::**AVLTree** ( ) `[inline]`

Default constructor.

Sets the number of elements to 0 and the root to NULL.

template<class dataType > virtual **os::AVLTree**< dataType >::~**AVLTree** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. The AVL tree must explicitly force deletion through the **AVL↩ Node**<**dataType**>**::remove()** (p. 346) function.

12.6.3 Member Function Documentation

template<class dataType > bool **os::AVLTree**< dataType >::balance (**smart_ptr**< **AVLNode**< dataType > > x) [inline], [protected]

Balances a single node.

Parameters

in	*x*	Node to be balanced

Returns

true if the node is already balanced, else, false

template<class dataType > bool **os::AVLTree**< dataType >::balanceDelete (**smart_ptr**< **AVLNode**< dataType > > x) [inline], [protected]

Removes a node and balances the tree.

Must receive as an argument a node in the tree. This function removes the node from the tree and re-balances the tree.

Parameters

in	*x*	Node to be deleted

Returns

true if successful, false if failed

template<class dataType > void **os::AVLTree**< dataType >::balanceUp (**smart_ptr**< **AVLNode**< dataType > > x) [inline], [protected]

Balances this node and ancestor nodes.

Balances the current node then orders it's parent node to be balanced as well. This process continues until a node has no parent (indicating the node is the root)

Parameters

in	*x*	Node to be balanced

Returns

void

template<class dataType > bool **os::AVLTree**< dataType >::checkBalance (**smart_ptr**< **AVLNode**< dataType > > x) [inline], [protected]

Checks if a sub-tree is balanced.

Checks if the received node is balanced. This operation is inexpensive as it merely involves comparing the heights of the children nodes.

Parameters

in	*x*	Node to be checked

Returns

true if balanced, false if not

template<class dataType > bool **os::AVLTree**< dataType >::doubleRotation (**smart_ptr**< **AVLNode**< dataType > > r, int dir) [inline], [protected]

Double-rotate a node.

Double-rotates a node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Node to be rotated
in	*dir*	Direction node is to be rotated

Returns

true if successful, else, false

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLTree**< dataType >::find (**smart_ptr**< dataType > x) [inline], [virtual]

Finds a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 326).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLTree**< dataType >::find (**smart_ptr**< **adnode**< dataType > > x) [inline]

Finds by adnode node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::adnode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLTree**< dataType >::find (**smart_ptr**< **AVLNode**< dataType > > x) [inline]

Finds by **AVLNode** (p. 343) node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::AVLNode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLTree**< dataType >::findBottom (**smart_ptr**< **AVLNode**< dataType > > x, int dir) [inline], [protected]

Find first or last node in a tree.

Finds the first or last node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Starting node
in	*dir*	Direction node to search in

Returns

First or last node in sub-tree

template<class dataType > bool **os::AVLTree**< dataType >::findDelete (**smart_ptr**< dataType > x) [inline], [virtual]

Finds and delete a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x dataType pointer to be compared against

Returns

true if the node was found and deleted, else false

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > bool **os::AVLTree**< dataType >::findDelete (**smart_ptr**< **AVLNode**< dataType > > x) [inline]

Finds and delete by node.

Finds a pointer to an object of type "dataType" given a comparison pointer to a node and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<data↩ Type> >). This function takes O(log(n)) where n is the number of elements in the tree and will re-balance the tree

[in] x os::AVLNode<dataType> pointer to be compared against

Returns

true if the node was found and deleted, else false

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLTree**< dataType >::getFirst ( ) [inline], [virtual]

Returns the first node.

For the AVL tree, the first node is defined as the child at index 1. Note that while an os↩ ::adnode<dataType> is returned, the true type of the pointer returned is os::AVLNode<dataType>. This function is O(log(n)).

Returns

The first node, if it exists

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::AVLTree**< dataType >::getLast ( ) [inline], [virtual]

Returns the last node.

For the AVL tree, the last node is defined as the child at index 0. Note that while an os↩ ::adnode<dataType> is returned, the true type of the pointer returned is os::AVLNode<dataType>. This function is O(log(n)).

Returns

The last node, if it exists

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLTree**< dataType >::getRoot ( ) [inline]

Return the root of the tree.

Returns

os::AVLTree<**dataType**>**::root** (p. 356)

template<class dataType > bool **os::AVLTree**< dataType >::insert (**smart_ptr**< **ads**< dataType > > x) [inline], [virtual]

Inserts an os::ads<dataType>

Inserts every element in a given abstract datastructure into this tree. Adopts the insertion function of os::ads<dataType>

[in] x pointer to os::ads<dataType>

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 328).

template<class dataType > bool **os::AVLTree**< dataType >::insert (**smart_ptr**< dataType > x) [inline], [virtual]

Inserts a data node.

Inserts a pointer to an object of type "dataType." This insertion will place the node into the binary tree and balance the tree. This function takes O(log(n)) where n is the number of elements in the tree.

[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > bool **os::AVLTree**< dataType >::singleRotation (**smart_ptr**< **AVLNode**< dataType > > r, int dir) [inline], [protected]

Rotates a node.

Rotates a node based on the dir argument provided. Note that 0 and 1 are the only valid directions.

Parameters

in	*x*	Node to be rotated
in	*dir*	Direction node is to be rotated

Returns

true if successful, else, false

template<class dataType > virtual unsigned int **os::AVLTree**< dataType >::size ( ) const [inline], [virtual]

Finds and delete a matching node.

Returns

os::AVLTree<**dataType**>**::numElements** (p. 356)

Reimplemented from **os::ads**< **dataType** > (p. 329).

12.6.4 Member Data Documentation

template<class dataType > unsigned int **os::AVLTree**< dataType >::numElements `[protected]`

Number of elements in the tree.

template<class dataType > **smart_ptr**<**AVLNode**<dataType> > **os::AVLTree**< dataType >::root `[protected]`

Root node of the tree.

12.7 os::constantPrinter Class Reference

Prints constant arrays to files.

Public Member Functions

- **constantPrinter** (std::string fileName, bool has_cpp=false)
  Single constructor.
- virtual ~**constantPrinter** ()
  Virtual destructor.
- void **addInclude** (std::string includeName)
  Add include file.
- void **addNamespace** (std::string namesp)
  Add a namespace.
- void **removeNamespace** ()
  Remove namespace.
- void **addComment** (std::string comment)
  Insert a comment.
- bool **hasCPP** () const
  Returns if the object is writing to a .cpp file.
- bool **good** () const
  Checks file status.
- void **addArray** (std::string name, uint32_t ∗arr, unsigned int length)
  Add a uin32_t∗ array.

Private Member Functions

- std::string **capitalize** (std::string str) const
  Capitalizes the string argument.
- std::string **tabs** () const
  Returns current tab depth.

Private Attributes

- std::ofstream **hFile**

  Output file for the .h file.
- std::ofstream **cppFile**

  Output file for the .cpp file.
- bool **_has_cpp**

  Holds if the object is generating a .cpp.
- unsigned int **namespaceDepth**

  Current namespace depth.

12.7.1 Detailed Description

Prints constant arrays to files.

This class outputs configured and populated constant arrays into .h and .cpp files, depending on the configuration. This class is meant to be used as a tool for automatically generating source code files.

12.7.2 Constructor & Destructor Documentation

os::constantPrinter::constantPrinter (std::string fileName, bool has_cpp = `false`)

Single constructor.

Creates a file of "filename.h" and, if has_cpp is set to "true," "filename.cpp" with appropriate include guards and a comment indicating the source of the file.

Parameters

`in`	*fileName*	String representing the file name
`in`	*has_cpp*	Optional boolean defining if a .cpp will be written

virtual os::constantPrinter::~constantPrinter ( ) `[virtual]`

Virtual destructor.

Closes all namespaces and #ifdefs, closes the .h file and .cpp if appropriate.

12.7.3 Member Function Documentation

void os::constantPrinter::addArray (std::string name, uint32_t * arr, unsigned int length)

Add a uin32_t* array.

Added an unsigned 32 bit integer array to the .h and .cpp file. Note that this array will be declared as constant.

Parameters

`in`	*arr*	Array to be written to the files

Parameters

in	*length*	Length of the received array

Returns

void

void os::constantPrinter::addComment (std::string comment)

Insert a comment.

Adds a comment. If the comment is a single line, '//' will be used, otherwise, a standard multi-line comment format will be used.

Parameters

in	*comment*	Comment string to be added as a comment

Returns

void

void os::constantPrinter::addInclude (std::string includeName)

Add include file.

Prints out "#include includeName" to the .h file. Since the .cpp file includes the .h file, it will include all of the .h file's includes

Parameters

in	*includeName*	Name of header file to be included

Returns

void

void os::constantPrinter::addNamespace (std::string namesp)

Add a namespace.

Adds a new namespace. Namespaces nest, so this function increments **constantPrinter↩ ::namespaceDepth** (p. 360). Both the .h and .cpp file have this namespace added.

Parameters

in	*namesp*	Namespace added to the file

Returns

void

std::string os::constantPrinter::capitalize (std::string str) const [private]

Capitalizes the string argument.

Primarily used for #ifdef and #define include guards, this function returns the string it is passed but with every single letter capitalized.

Parameters

in	*str*	String to be capitalized

Returns

std::string with each letter capitalized

bool os::constantPrinter::good ( ) const [inline]

Checks file status.

Checks to ensure that both the .h and .cpp file can be written to. Will not consider the .cpp file if the .cpp file is not being written to.

Returns

file status

bool os::constantPrinter::hasCPP ( ) const [inline]

Returns if the object is writing to a .cpp file.

Returns

constantPrinter::_has_cpp (p. 360)

void os::constantPrinter::removeNamespace ( )

Remove namespace.

Ends the current namespace with a '}' in both the .h and .cpp file. Decrements **constantPrinter::namespaceDepth** (p. 360).

Returns

void

std::string os::constantPrinter::tabs ( ) const [private]

Returns current tab depth.

Again used to streamline large projects. This function returns an std::string with tab characters equal to the current number of nested namespaces.

Returns

std::string containing **os::constantPrinter::namespaceDepth** (p. 360) tabs

12.7.4 Member Data Documentation

bool os::constantPrinter::_has_cpp `[private]`

Holds if the object is generating a .cpp.

std::ofstream os::constantPrinter::cppFile `[private]`

Output file for the .cpp file.

std::ofstream os::constantPrinter::hFile `[private]`

Output file for the .h file.

unsigned int os::constantPrinter::namespaceDepth `[private]`

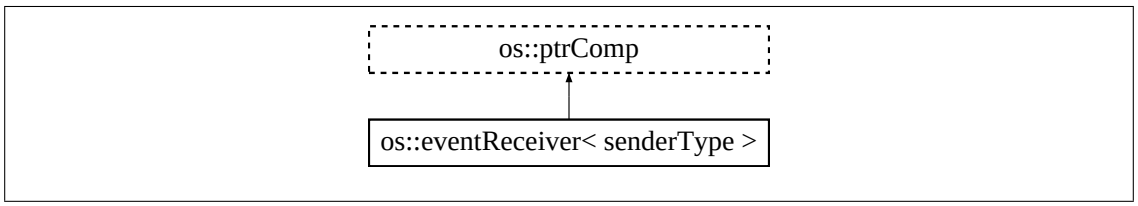
Current namespace depth.

In order to streamline large projects, arrays of constants should be placed inside namespaces. This variable allows for the creation and management of nested namespaces.

12.8 os::eventReceiver< senderType > Class Template Reference

Class which enables event receiving.

Inheritance diagram for os::eventReceiver< senderType >:



Public Member Functions

- **eventReceiver** ()

  Default constructor.
- virtual ~**eventReceiver** ()

  Virtual destructor.
- void **pushSender** (**smart_ptr**< senderType > ptr)

  Add a sender to the list.
- void **removeSender** (**smart_ptr**< senderType > ptr)

  Remove sender from the sender list.

Private Member Functions

- virtual void **receiveEvent** (**smart_ptr**< senderType > src)

  Receive event notification.

Private Attributes

- **asyncAVLTree**< senderType > **senders**

  List of sender.

Friends

- template<typename receiverType >
  class **eventSender**

12.8.1 Detailed Description

template<class senderType>
class os::eventReceiver< senderType >

Class which enables event receiving.

Each receiver contains a list of senders. When the receiver is destroyed, it removes itself from all senders to which it is registered.

12.8.2 Constructor & Destructor Documentation

template<class senderType > **os::eventReceiver**< senderType >::**eventReceiver** ( ) `[inline]`

Default constructor.

The default constructor for the smart set configures the only data type in this class properly. No additional constructor arguments are required.

template<class senderType > virtual **os::eventReceiver**< senderType >::~**eventReceiver** ( ) `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.8.3 Member Function Documentation

template<class senderType > void **os::eventReceiver**< senderType >::pushSender (**smart_ptr**< senderType > ptr)

Add a sender to the list.

Adds a sender of the sender type expected by this receiver type. Note that the sender type is expected to inherit from **os::eventSender** (p. 363).

Parameters

ptr	Sender to be added to the set

Returns

void

template<class senderType > virtual void **os::eventReceiver**< senderType >::receiveEvent (**smart_ptr**< senderType > src) [inline], [private], [virtual]

Receive event notification.

This function is meant to be reimplemented by all event receivers to do some action on the event.

Parameters

src	The source of the event

Returns

void

template<class senderType > void **os::eventReceiver**< senderType >::removeSender (**smart_ptr**< senderType > ptr)

Remove sender from the sender list.

Removes a sender from the sender list. Note that this also removes this receiver from the receiver list of the sender which it is passed.

Parameters

ptr	Sender to be removed to the set

Returns

void

12.8.4 Friends And Related Function Documentation

template<class senderType > template<typename receiverType > friend class **eventSender** [friend]

The sender must be able to remove itself from the private senders list inside the event receiver. Additionally, the sender must be able to send an event to the receiver.

12.8.5 Member Data Documentation

template<class senderType > **asyncAVLTree**<senderType> **os::eventReceiver**< senderType >::senders `[private]`

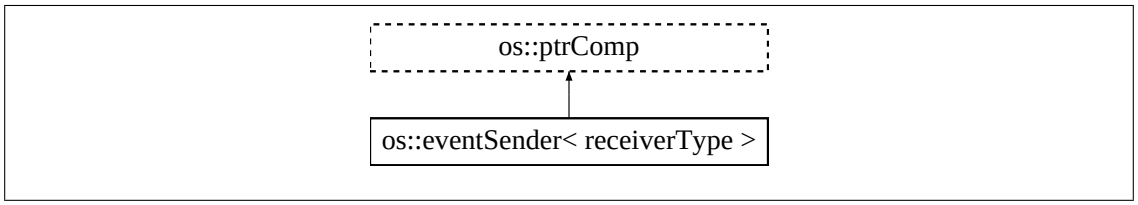
List of sender.

When the receiver is destroyed, this list is used to remove itself from all its senders.

12.9 os::eventSender< receiverType > Class Template Reference

Class which enables event sending.

Inheritance diagram for os::eventSender< receiverType >:



Public Member Functions

- **eventSender** ()

  Default constructor.
- virtual ~**eventSender** ()

  Virtual destructor.
- void **pushReceivers** (**smart_ptr**< receiverType > ptr)

  Add a receiver to the list.
- void **removeReceivers** (**smart_ptr**< receiverType > ptr)

  Remove receiver from the receiver list.

Protected Member Functions

- virtual void **sendEvent** (**smart_ptr**< receiverType > ptr)

  Receive event notification.
- void **triggerEvent** ()

  Sends an event to all receivers.

Private Attributes

- **asyncAVLTree**< receiverType > **receivers**

  List of receivers.

Friends

- template<typename senderType >
  class **eventReceiver**

12.9.1 Detailed Description

template<class receiverType>
class os::eventSender< receiverType >

Class which enables event sending.

Each sender contains a list of receivers. When an event is triggered, the sender iterates through the list to send the event to all receivers.

12.9.2 Constructor & Destructor Documentation

template<class receiverType > **os::eventSender**< receiverType >::**eventSender** ( ) `[inline]`

Default constructor.

The default constructor for the smart set configures the only data type in this class properly. No additional constructor arguments are required.

template<class receiverType > virtual **os::eventSender**< receiverType >::~**eventSender** ( )
`[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.9.3 Member Function Documentation

template<class receiverType > void **os::eventSender**< receiverType >::pushReceivers (
smart_ptr< receiverType > ptr)

Add a receiver to the list.

Adds a receiver of the receiver type expected by this sender type. Note that the receiver type is expected to inherit from **os::eventReceiver** (p. 360).

Parameters

ptr	Receiver to be added to the set

Returns

void

template<class receiverType > void **os::eventSender**< receiverType >::removeReceivers (
smart_ptr< receiverType > ptr)

Remove receiver from the receiver list.

Removes a receiver from the receiver list. Note that this also removes this sender from the sender list of the receiver which it is passed.

Parameters

ptr	Receiver to be removed to the set

Returns

void

template<class receiverType > virtual void **os::eventSender**< receiverType >::sendEvent (**smart_ptr**< receiverType > ptr) [protected], [virtual]

Receive event notification.

This function can be re-implemented by event senders. This function allows some function other than "receiveEvent" to be sent by the event sender to an event receiver.

Parameters

ptr	The target of the event

Returns

void

template<class receiverType > void **os::eventSender**< receiverType >::triggerEvent ( ) [protected]

Sends an event to all receivers.

Iterates through the set of receivers and sends an event to each one. This calls the **os::event↩ Sender**<**receiverType**>**::sendEvent** (p. 365) function with each receiver as an argument.

Returns

void

12.9.4 Friends And Related Function Documentation

template<class receiverType > template<typename senderType > friend class **eventReceiver** [friend]

The receiver must be able to remove itself from the private receivers list inside the event sender.

12.9.5 Member Data Documentation

template<class receiverType > **asyncAVLTree**<receiverType> **os::eventSender**< receiverType >::receivers [private]

List of receivers.

This list is used to send events to all receivers. When the sender is destroyed, it must remove itself from all its receivers.

12.10 os::indirectMatrix< dataType > Class Template Reference

Indirect matrix.

Public Member Functions

- **indirectMatrix** (uint32_t w=0, uint32_t h=0)

  Default constructor.
- **indirectMatrix** (const **matrix**< dataType > &m)

  Copy constructor.
- **indirectMatrix** (const **indirectMatrix**< dataType > &m)

  Copy constructor.
- **indirectMatrix** (const **smart_ptr**< dataType > d, uint32_t w, uint32_t h)

  Data array constructor.
- **indirectMatrix** (**smart_ptr**< **smart_ptr**< dataType > > d, uint32_t w, uint32_t h)

  Indirect data array constructor.
- virtual ~**indirectMatrix** ()

  Virtual destructor.
- **indirectMatrix**< dataType > & **operator=** (const **matrix**< dataType > &m)

  Equality constructor.
- **indirectMatrix**< dataType > & **operator=** (const **indirectMatrix**< dataType > &m)

  Equality constructor.
- **smart_ptr**< dataType > & **get** (uint32_t w, uint32_t h)

  Return pointer to a matrix element.
- const **smart_ptr**< dataType > & **constGet** (uint32_t w, uint32_t h) const

  Return constant pointer to a matrix element.
- **smart_ptr**< dataType > & **operator()** (uint32_t w, uint32_t h)

  Return pointer to a matrix element.
- **smart_ptr**< **smart_ptr**< dataType > > **getArray** ()

  Return pointer to the pointer array.
- const **smart_ptr**< **smart_ptr**< dataType > > **getConstArray** () const

  Return a constant pointer to the pointer array.
- uint32_t **getWidth** () const

  Return width of matrix.
- uint32_t **getHeight** () const

  Return height of matrix.

Private Attributes

- uint32_t **width**

  Width of the matrix.
- uint32_t **height**

  Height of the matrix.
- **smart_ptr**< **smart_ptr**< dataType > > **data**

  Data array pointers.

Friends

- class **matrix**< **dataType** >

  Raw matrix interacting with indirect matrix.

12.10.1 Detailed Description

template<class dataType>
class os::indirectMatrix< dataType >

Indirect matrix.

This matrix class contains an array to pointers of the data type. It can interact with os::matrix<data↩Type>.

12.10.2 Constructor & Destructor Documentation

template<class dataType> **os::indirectMatrix**< dataType >::**indirectMatrix** (uint32_t w = `0`, uint32_t h = `0`)

Default constructor.

Constructs array of size w∗h and sets all of the data to 0. If no width and height are provided, the data array is not initialized.

Parameters

`in`	*w*	Width of matrix, default 0
`in`	*h*	Height of matrix, default 0

template<class dataType> **os::indirectMatrix**< dataType >::**indirectMatrix** (const **matrix**< dataType > & m)

Copy constructor.

Constructs a new indirect matrix from the given raw matrix. The indirect matrix converts the array of object to an array of pointers.

Parameters

`in`	*m*	Indirect matrix to be copied

template<class dataType> **os::indirectMatrix**< dataType >::**indirectMatrix** (const **indirectMatrix**< dataType > & m)

Copy constructor.

Constructs a new indirect matrix from the given indirect matrix. The two indirect matrices do not share data array, the new indirect matrix builds its own array.

Parameters

in	*m*	Indirect matrix to be copied

template<class dataType> **os::indirectMatrix**< dataType >::**indirectMatrix** (const **smart_ptr**< dataType > d, uint32_t w, uint32_t h)

Data array constructor.

Constructs a new indirect matrix from an array of the correct data type. This constructor will build an new indirect array based on the specified size.

Parameters

in	*d*	Data array to be copied
in	*w*	Width of matrix
in	*d*	Height of matrix

template<class dataType> **os::indirectMatrix**< dataType >::**indirectMatrix** (**smart_ptr**< **smart_ptr**< dataType > > d, uint32_t w, uint32_t h)

Indirect data array constructor.

Constructs a new indirect matrix from an indirect array of the correct data type. This constructor will build an new indirect array based on the specified size.

Parameters

in	*d*	Indirect data array to be copied
in	*w*	Width of matrix
in	*d*	Height of matrix

template<class dataType> virtual **os::indirectMatrix**< dataType >::~**indirectMatrix** ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.10.3 Member Function Documentation

template<class dataType> const **smart_ptr**<dataType>& **os::indirectMatrix**< dataType >::constGet (uint32_t w, uint32_t h) const

Return constant pointer to a matrix element.

Uses a width and height position to index an element of the array. This function returns a constant reference, meaning changes cannot be made to the matrix.

Parameters

in	*w*	X position
in	*h*	Y position

Returns

Constant reference to matrix element pointer

template<class dataType> **smart_ptr**<dataType>& **os::indirectMatrix**< dataType >::get (uint32_t w, uint32_t h)

Return pointer to a matrix element.

Uses a width and height position to index an element of the array. This function returns a reference, allowing for changes to be made to the matrix.

Parameters

in	*w*	X position
in	*h*	Y position

Returns

Modifiable reference to matrix element pointer

template<class dataType> **smart_ptr**<**smart_ptr**<dataType> > **os::indirectMatrix**< dataType >::getArray ( ) [inline]

Return pointer to the pointer array.

The array which is returned allows for modification of the array. It is up to functions using this array to ensure the integrity of the indirect matrix.

Returns

os::indirectMatrix<**dataType**>**::data** (p. 371)

template<class dataType> const **smart_ptr**<**smart_ptr**<dataType> > **os::indirectMatrix**< dataType >::getConstArray ( ) const [inline]

Return a constant pointer to the pointer array.

The array which is returned allows for access to the array. The provided array may not be modified.

Returns

os::indirectMatrix<**dataType**>**::data** (p. 371)

template<class dataType> uint32_t **os::indirectMatrix**< dataType >::getHeight ( ) const [inline]

Return height of matrix.

Returns

indirectMatrix<**dataType**>**::height** (p. 371)

template<class dataType> uint32_t **os::indirectMatrix**< dataType >::getWidth ( ) const `[inline]`

Return width of matrix.

Returns

indirectMatrix<**dataType**>**::width** (p. 371)

template<class dataType> **smart_ptr**<dataType>& **os::indirectMatrix**< dataType >::operator() (uint32_t w, uint32_t h) `[inline]`

Return pointer to a matrix element.

Uses a width and height position to index an element of the array. This function returns a reference, allowing for changes to be made to the matrix.

Parameters

`in`	*w*	X position
`in`	*h*	Y position

Returns

Modifiable reference to matrix element pointer

template<class dataType> **indirectMatrix**<dataType>& **os::indirectMatrix**< dataType >::operator= (const **matrix**< dataType > & m)

Equality constructor.

Re-constructs the indirect matrix from a raw matrix. Note that the two matrices do not share the same data array.

Parameters

`in`	*m*	Reference to matrix being copied

Returns

Reference to self

template<class dataType> **indirectMatrix**<dataType>& **os::indirectMatrix**< dataType >::operator= (const **indirectMatrix**< dataType > & m)

Equality constructor.

Re-constructs the indirect matrix from another indirect matrix. Note that the two matrices do not share the same data array.

Parameters

in	*m*	Reference to matrix being copied

Returns

Reference to self

12.10.4 Friends And Related Function Documentation

template<class dataType> friend class **matrix**< dataType > [friend]

Raw matrix interacting with indirect matrix.

The os::matrix<dataType> class must be able to access the size and data of the indirect matrix because and raw matrix can be constructed from an indirect matrix.

12.10.5 Member Data Documentation

template<class dataType> **smart_ptr**<**smart_ptr**<dataType> > **os::indirectMatrix**< dataType >::data [private]

Data array pointers.

For the indirect matrix class, this array contains pointers to all of the data used by the matrix in a block of size width∗height.

template<class dataType> uint32_t **os::indirectMatrix**< dataType >::height [private]

Height of the matrix.

template<class dataType> uint32_t **os::indirectMatrix**< dataType >::width [private]

Width of the matrix.

12.11 os::matrix< dataType > Class Template Reference

Raw matrix.

Public Member Functions

- **matrix** (uint32_t w=0, uint32_t h=0)

  Default constructor.
- **matrix** (const **matrix**< dataType > &m)

  Copy constructor.
- **matrix** (const **indirectMatrix**< dataType > &m)

  Copy constructor.
- **matrix** (const **smart_ptr**< dataType > d, uint32_t w, uint32_t h)

  Data array constructor.

- **matrix** (**smart_ptr**< **smart_ptr**< dataType > > d, uint32_t w, uint32_t h)

  Indirect data array constructor.
- virtual ~**matrix** ()

  Virtual destructor.
- **matrix**< dataType > & **operator=** (const **matrix**< dataType > &m)

  Equality constructor.
- **matrix**< dataType > & **operator=** (const **indirectMatrix**< dataType > &m)

  Equality constructor.
- dataType & **get** (uint32_t w, uint32_t h)

  Return matrix element.
- const dataType & **constGet** (uint32_t w, uint32_t h) const

  Return constant matrix element.
- dataType & **operator()** (uint32_t w, uint32_t h)

  Return matrix element.
- **smart_ptr**< dataType > **getArray** ()

  Return pointer to the array.
- const **smart_ptr**< dataType > **getConstArray** () const

  Return a constant pointer to the array.
- uint32_t **getWidth** () const

  Return width of matrix.
- uint32_t **getHeight** () const

  Return height of matrix.

Private Attributes

- uint32_t **width**

  Width of the matrix.
- uint32_t **height**

  Height of the matrix.
- **smart_ptr**< dataType > **data**

  Data array.

Friends

- class **indirectMatrix**< **dataType** >

  Indirect matrix interacting with raw matrix.

12.11.1 Detailed Description

template<class dataType>
class os::matrix< dataType >

Raw matrix.

This matrix class contains an array of the data type. It can interact with os::indirectMatrix<data↩ Type>.

12.11.2 Constructor & Destructor Documentation

template<class dataType> **os::matrix**< dataType >::**matrix** (uint32_t w = 0, uint32_t h = 0)

Default constructor.

Constructs array of size w∗h and sets all of the data to 0. If no width and height are provided, the data array is not initialized.

Parameters

in	*w*	Width of matrix, default 0
in	*h*	Height of matrix, default 0

template<class dataType> **os::matrix**< dataType >::**matrix** (const **matrix**< dataType > & m)

Copy constructor.

Constructs a new raw matrix from the given raw matrix. The two matrices do not share the same data array.

Parameters

in	*m*	Matrix to be copied

template<class dataType> **os::matrix**< dataType >::**matrix** (const **indirectMatrix**< dataType > & m)

Copy constructor.

Constructs a new raw matrix from the given indirect matrix. The raw matrix converts the array of pointers to an array of objects

Parameters

in	*m*	Indirect matrix to be copied

template<class dataType> **os::matrix**< dataType >::**matrix** (const **smart_ptr**< dataType > d, uint32_t w, uint32_t h)

Data array constructor.

Constructs a new raw matrix from an array of the correct data type. This constructor will build an new array based on the specified size.

Parameters

in	*d*	Data array to be copied
in	*w*	Width of matrix
in	*d*	Height of matrix

template<class dataType> **os::matrix**< dataType >::**matrix** (**smart_ptr**< **smart_ptr**< dataType > > d, uint32_t w, uint32_t h)

Indirect data array constructor.

Constructs a new raw matrix from an indirect array of the correct data type. This constructor will build an new array based on the specified size.

Parameters

in	*d*	Indirect data array to be copied
in	*w*	Width of matrix
in	*d*	Height of matrix

template<class dataType> virtual **os::matrix**< dataType >::~**matrix** ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.11.3 Member Function Documentation

template<class dataType> const dataType& **os::matrix**< dataType >::constGet (uint32_t w, uint32_t h) const

Return constant matrix element.

Uses a width and height position to index an element of the array. This function returns a constant reference, meaning changes cannot be made to the matrix.

Parameters

in	*w*	X position
in	*h*	Y position

Returns

Constant reference to matrix element

template<class dataType> dataType& **os::matrix**< dataType >::get (uint32_t w, uint32_t h)

Return matrix element.

Uses a width and height position to index an element of the array. This function returns a reference, allowing for changes to be made to the matrix.

Parameters

in	*w*	X position
in	*h*	Y position

Returns

Modifiable reference to matrix element

template<class dataType> **smart_ptr**<dataType> **os::matrix**< dataType >::getArray ( ) `[inline]`

Return pointer to the array.

The array which is returned allows for modification of the array. It is up to functions using this array to ensure the integrity of the matrix.

Returns

os::matrix<**dataType**>**::data** (p. 376)

template<class dataType> const **smart_ptr**<dataType> **os::matrix**< dataType >::getConstArray ( ) const `[inline]`

Return a constant pointer to the array.

The array which is returned allows for access to the array. The provided array may not be modified.

Returns

os::matrix<**dataType**>**::data** (p. 376)

template<class dataType> uint32_t **os::matrix**< dataType >::getHeight ( ) const `[inline]`

Return height of matrix.

Returns

matrix<**dataType**>**::height** (p. 377)

template<class dataType> uint32_t **os::matrix**< dataType >::getWidth ( ) const `[inline]`

Return width of matrix.

Returns

matrix<**dataType**>**::width** (p. 377)

template<class dataType> dataType& **os::matrix**< dataType >::operator() (uint32_t w, uint32_t h) `[inline]`

Return matrix element.

Uses a width and height position to index an element of the array. This function returns a reference, allowing for changes to be made to the matrix.

Parameters

`in`	*w*	X position
`in`	*h*	Y position

Returns

Modifiable reference to matrix element

template<class dataType> **matrix**<dataType>& **os::matrix**< dataType >::operator= (const **matrix**< dataType > & m)

Equality constructor.

Re-constructs the raw matrix from another raw matrix. Note that the two matrices do not share the same data array.

Parameters

in	*m*	Reference to matrix being copied

Returns

Reference to self

template<class dataType> **matrix**<dataType>& **os::matrix**< dataType >::operator= (const **indirectMatrix**< dataType > & m)

Equality constructor.

Re-constructs the raw matrix from an indirect matrix. Note that the two matrices do not share the same data array.

Parameters

in	*m*	Reference to matrix being copied

Returns

Reference to self

12.11.4 Friends And Related Function Documentation

template<class dataType> friend class **indirectMatrix**< dataType > [friend]

Indirect matrix interacting with raw matrix.

The os::indirectMatrix<dataType> class must be able to access the size and data of the raw matrix because and indirect matrix can be constructed from a raw matrix.

12.11.5 Member Data Documentation

template<class dataType> **smart_ptr**<dataType> **os::matrix**< dataType >::data [private]

Data array.

For the raw matrix class, this array contains all of the data used by the matrix in a block of size width∗height.

template<class dataType> uint32_t **os::matrix**< dataType >::height `[private]`

Height of the matrix.

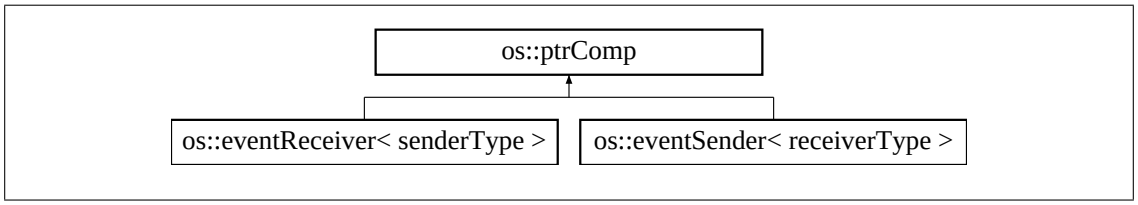
template<class dataType> uint32_t **os::matrix**< dataType >::width `[private]`

Width of the matrix.

12.12 os::ptrComp Class Reference

Pointer compare interface.

Inheritance diagram for os::ptrComp:



Public Member Functions

- virtual ~**ptrComp** ()

  Virtual destructor.
- virtual bool **operator==** (const **ptrComp** &l) const

  Equality test.
- virtual bool **operator**> (const **ptrComp** &l) const

  Greater than test.
- virtual bool **operator**< (const **ptrComp** &l) const

  Less than test.
- virtual bool **operator**>**=** (const **ptrComp** &l) const

  Greater than/equal to test.
- virtual bool **operator**<**=** (const **ptrComp** &l) const

  Less than/equal to test.

12.12.1 Detailed Description

Pointer compare interface.

Allows a class which does not define comparison operators to be placed into an abstract datastructure by defining comparison to be address comparison.

12.12.2 Constructor & Destructor Documentation

virtual os::ptrComp::~ptrComp ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.12.3 Member Function Documentation

virtual bool os::ptrComp::operator< (const **ptrComp** & l) const [inline], [virtual]

Less than test.

virtual bool os::ptrComp::operator<= (const **ptrComp** & l) const [inline], [virtual]

Less than/equal to test.

virtual bool os::ptrComp::operator== (const **ptrComp** & l) const [inline], [virtual]

Equality test.

virtual bool os::ptrComp::operator> (const **ptrComp** & l) const [inline], [virtual]

Greater than test.

virtual bool os::ptrComp::operator>= (const **ptrComp** & l) const [inline], [virtual]

Greater than/equal to test.

12.13 os::smart_ptr< dataType > Class Template Reference

Reference counted pointer.

Public Member Functions

- **smart_ptr** ()

  Default constructor.
- **smart_ptr** (const **smart_pointer_type** t, const std::atomic< unsigned long > *rc, const dataType *rp, const **void_rec** f)

  Forced constructor.
- **smart_ptr** (const **smart_ptr**< dataType > &sp)

  Copy constructor.
- **smart_ptr** (const dataType *rp, **smart_pointer_type** typ=**raw_type**)

  Standard constructor.
- **smart_ptr** (const dataType *rp, const **void_rec** destructor)

  Dynamic deletion constructor.
- virtual ~**smart_ptr** ()

Virtual destructor.

- **smart_ptr** (const int rp)

  Integer constructor.
- **smart_ptr** (const long rp)

  Long constructor.
- **smart_ptr** (const unsigned long rp)

  Unsigned long constructor.
- **smart_pointer_type getType** () const

  Return type.
- dataType ∗ **get** ()

  Return data.
- const dataType ∗ **get** () const

  Return constant data.
- const dataType ∗ **constGet** () const

  Return constant data.
- const std::atomic< unsigned long > ∗ **getRefCount** () const

  Return constant reference count.
- **void_rec getFunc** () const

  Return deletion function.
- bool **operator!** () const

  Inverted boolean conversion.
- **operator bool** () const

  Boolean conversion.
- dataType & **operator**∗ ()

  De-reference conversion.
- const dataType & **operator**∗ () const

  Constant de-reference conversion.
- dataType ∗ **operator->** ()

  Pointer pass.
- const dataType ∗ **operator->** () const

  Constant pointer pass.
- dataType & **operator[ ]** (unsigned int i)

  Array de-reference.
- const dataType & **operator[ ]** (unsigned int i) const

  Constant array de-reference.
- **smart_ptr**< dataType > & **bind** (**smart_ptr**< dataType > sp)

  Bind copy.
- **smart_ptr**< dataType > & **bind** (const dataType ∗rp)

  Bind raw copy.
- **smart_ptr**< dataType > & **operator=** (const **smart_ptr**< dataType > source)

  Equals copy.
- **smart_ptr**< dataType > & **operator=** (const dataType ∗source)

  Bind raw copy.

- **smart_ptr**< dataType > & **operator=** (const int source)

  Bind integer copy.
- **smart_ptr**< dataType > & **operator=** (const long source)

  Bind long copy.
- **smart_ptr**< dataType > & **operator=** (const unsigned long source)

  Bind unsigned long copy.
- int **compare** (const **smart_ptr**< dataType > &c) const

  *Compare **os::smart_ptr** (p. 378).*
- int **compare** (const dataType ∗c) const

  Compare raw pointers.
- int **compare** (const unsigned long c) const

  Compare cast long.

Private Member Functions

- void **teardown** ()

  Delete data.

Private Attributes

- **smart_pointer_type type**

  Stores the type.
- std::atomic< unsigned long > ∗ **ref_count**

  Reference count.
- dataType ∗ **raw_ptr**

  Pointer to data.
- **void_rec func**

  Non-standard deletion.

12.13.1 Detailed Description

template<class dataType>
class os::smart_ptr< dataType >

Reference counted pointer.

The **os::smart_ptr** (p. 378) template class allows for automatic memory management. **os::smart_ptr** (p. 378)'s have a type defined by **os::smart_pointer_type** (p. 316) which defines the copy and deletion behaviour of the object.

12.13.2 Constructor & Destructor Documentation

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** ( ) `[inline]`

Default constructor.

Constructs an **os::smart_ptr** (p. 378) of type **os::null_type** (p. 316). All private data is set to 0 or NULL.

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const **smart_pointer_type** t, const std::atomic< unsigned long > * rc, const dataType * rp, const **void_rec** f) [inline]

Forced constructor.

Constructs an **os::smart_ptr** (p. 378) explicitly from each of the parameters provided. This constructor is primarily used for testing purposes.

Parameters

in	*t*	Type definition for the object
in,out	*rp*	Pointer to the reference count
in	*rp*	Raw pointer object is managing
in	*f*	Dynamic deletion function

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const **smart_ptr**< dataType > & sp) [inline]

Copy constructor.

Constructs an **os::smart_ptr** (p. 378) from an existing **os::smart_ptr** (p. 378). Will increment the reference count as defined by the received **os::smart_pointer_type** (p. 316).

Parameters

in,out	*sp*	Reference to data being copied

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const dataType * rp, **smart_pointer_type** typ = **raw_type**) [inline]

Standard constructor.

Constructs an **os::smart_ptr** (p. 378) from a raw pointer and a type. This is the most commonly used **os::smart_ptr** (p. 378) constructor, other than the copy constructor. Note that **os::shared_type_dynamic_delete** (p. 316) cannot be constructed through this method.

Parameters

in	*rp*	Raw pointer object is managing
in	*typ*	Defines reference count behaviour

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const dataType * rp, const **void_rec** destructor) [inline]

Dynamic deletion constructor.

Constructs an **os::smart_ptr** (p. 378) from a raw pointer and a destruction function. This constructor generates an **os::smart_ptr** (p. 378) of type **os::shared_type_dynamic_delete** (p. 316).

Parameters

in	*rp*	Raw pointer object is managing
in	*destructor*	Defines the function to be executed on destroy

template<class dataType> virtual **os::smart_ptr**< dataType >::~**smart_ptr** ( ) [inline], [virtual]

Virtual destructor.

Calls **os::smart_ptr**<**dataType**>**::teardown()** (p. 388) before destroying the object.

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const int rp) [inline]

Integer constructor.

Constructs an **os::smart_ptr** (p. 378) from an integer. The assumption is that this integer is 0 (or NULL). This function is still legal if the integer is not NULL, this allows for casting, although such usage is discouraged.

Parameters

in	*rp*	Integer cast to raw pointer

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const long rp) [inline]

Long constructor.

Constructs an **os::smart_ptr** (p. 378) from an long. The assumption is that this long is 0 (or NULL). This function is still legal if the long is not NULL, this allows for casting, although such usage is discouraged.

Parameters

in	*rp*	Long cast to raw pointer

template<class dataType> **os::smart_ptr**< dataType >::**smart_ptr** (const unsigned long rp) [inline]

Unsigned long constructor.

Constructs an **os::smart_ptr** (p. 378) from an unsigned long. The assumption is that this unsigned long is 0 (or NULL). This function is still legal if the unsigned long is not NULL, this allows for casting, although such usage is discouraged.

Parameters

in	*rp*	Unsigned long cast to raw pointer

12.13.3 Member Function Documentation

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::bind (**smart_ptr**< dataType > sp) [inline]

Bind copy.

Binds to an **os::smart_ptr** (p. 378) from an existing **os::smart_ptr** (p. 378). Will increment the reference count as defined by the received **os::smart_pointer_type** (p. 316).

Parameters

in	*sp*	Reference to data being copied

Returns

Reference to self

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::bind (const dataType * rp) [inline]

Bind raw copy.

Binds to an **os::smart_ptr** (p. 378) from a dataType pointer. This new **os::smart_ptr** (p. 378) will be of type **os::raw_type** (p. 316) unless the dataType pointer is NULL, then it will be of type **os::null_type** (p. 316).

Parameters

in	*rp*	Reference to dataType pointer

Returns

Reference to self

template<class dataType> int **os::smart_ptr**< dataType >::compare (const **smart_ptr**< dataType > & c) const [inline]

Compare **os::smart_ptr** (p. 378).

Compares two pointers to the same type by address and returns the result in the form of a 1,0 or -1. Note that the **os::smart_ptr**<**dataType**>**::type** (p. 389) of the objects does not factor into this comparison.

Parameters

in	*c*	os::smart_ptr<dataType>

Returns

1, 0, -1 (Greater than, equal to, less than)

template<class dataType> int **os::smart_ptr**< dataType >::compare (const dataType * c) const [inline]

Compare raw pointers.

Compares a os::smart_ptr<dataType> and a raw pointer of type dataType and returns the result in the form of a 1,0 or -1.

Parameters

in	*c*	Raw dataType pointer

Returns

1, 0, -1 (Greater than, equal to, less than)

template<class dataType> int **os::smart_ptr**< dataType >::compare (const unsigned long c) const [inline]

Compare cast long.

Compares a os::smart_ptr<dataType> and an unsigned long, returning the result in the form of a 1,0 or -1.

Parameters

in	*c*	Unsigned long cast to dataType pointer

Returns

1, 0, -1 (Greater than, equal to, less than)

template<class dataType> const dataType* **os::smart_ptr**< dataType >::constGet ( ) const [inline]

Return constant data.

Returns the constant dataType pointer of the **os::smart_ptr** (p. 378).

Returns

dataType* in constant form, **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388)

template<class dataType> dataType* **os::smart_ptr**< dataType >::get ( ) [inline]

Return data.

Returns the dataType pointer of the **os::smart_ptr** (p. 378).

Returns

dataType* in modifiable form, **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388)

template<class dataType> const dataType* **os::smart_ptr**< dataType >::get ( ) const [inline]

Return constant data.

Returns the constant dataType pointer of the **os::smart_ptr** (p. 378).

Returns

dataType* in constant form, **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388)

template<class dataType> **void_rec os::smart_ptr**< dataType >::getFunc ( ) const [inline]

Return deletion function.

Returns the deletion function if it exists. (Note that the deletion function only exists in **os::shared_type_dynamic_delete** (p. 316) mode)

Returns

os::void_rec (p. 315) **os::smart_ptr**<**dataType**>**::func** (p. 388)

template<class dataType> const std::atomic<unsigned long>* **os::smart_ptr**< dataType >::getRefCount ( ) const [inline]

Return constant reference count.

Returns a constant pointer of the reference count.

Returns

unsigned long* in constant form, **os::smart_ptr**<**dataType**>**::ref_count** (p. 389)

template<class dataType> **smart_pointer_type os::smart_ptr**< dataType >::getType ( ) const [inline]

Return type.

Returns the **os::smart_pointer_type** (p. 316) of the **os::smart_ptr** (p. 378).

Returns

os::smart_pointer_type (p. 316) **os::smart_ptr**<**dataType**>**::type** (p. 389)

template<class dataType> **os::smart_ptr**< dataType >::operator bool ( ) const [inline]

Boolean conversion.

Returns

os::smart_ptr<**dataType**>**::raw_ptr** (p. 388) cast to boolean

template<class dataType> bool **os::smart_ptr**< dataType >::operator! ( ) const [inline]

Inverted boolean conversion.

Returns

Inverse of **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) cast to boolean

template<class dataType> dataType& **os::smart_ptr**< dataType >::operator* ( ) `[inline]`

De-reference conversion.

Returns

dataType reference of **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) de-referenced

template<class dataType> const dataType& **os::smart_ptr**< dataType >::operator* ( ) const `[inline]`

Constant de-reference conversion.

Returns

Constant dataType reference of **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) de-referenced

template<class dataType> dataType* **os::smart_ptr**< dataType >::operator-> ( ) `[inline]`

Pointer pass.

Returns

os::smart_ptr<**dataType**>**::raw_ptr** (p. 388)

template<class dataType> const dataType* **os::smart_ptr**< dataType >::operator-> ( ) const `[inline]`

Constant pointer pass.

Returns

Constant **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388)

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::operator= (const **smart_ptr**< dataType > source) `[inline]`

Equals copy.
Calls **os::smart_ptr**<**dataType**>**::bind** (p. 383).

Parameters

`in`	*source*	Reference to data being copied

Returns

Reference to self

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::operator= (const dataType * source) `[inline]`

Bind raw copy.
Calls **os::smart_ptr**<**dataType**>**::bind** (p. 383).

Parameters

in	*source*	Reference to dataType pointer

Returns

Reference to self

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::operator= (const int source) [inline]

Bind integer copy.

Calls **os::smart_ptr**<**dataType**>**::bind** (p. 383) with the integer cast to a dataType pointer.

Parameters

in	*source*	Integer cast to raw pointer

Returns

Reference to self

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::operator= (const long source) [inline]

Bind long copy.

Calls **os::smart_ptr**<**dataType**>**::bind** (p. 383) with the long cast to a dataType pointer.

Parameters

in	*source*	Long cast to raw pointer

Returns

Reference to self

template<class dataType> **smart_ptr**<dataType>& **os::smart_ptr**< dataType >::operator= (const unsigned long source) [inline]

Bind unsigned long copy.

Calls **os::smart_ptr**<**dataType**>**::bind** (p. 383) with the unsigned long cast to a dataType pointer.

Parameters

in	*source*	Unsigned long cast to raw pointer

Returns

Reference to self

template<class dataType> dataType& **os::smart_ptr**< dataType >::operator[] (unsigned int i) `[inline]`

Array de-reference.

Returns

dataType reference of **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) incremented i de-referenced

template<class dataType> const dataType& **os::smart_ptr**< dataType >::operator[] (unsigned int i) const `[inline]`

Constant array de-reference.

Returns

Constant dataType reference of **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) incremented i de-referenced

template<class dataType> void **os::smart_ptr**< dataType >::teardown ( ) `[inline]`, `[private]`

Delete data.

Tears down the **os::smart_ptr** (p. 378). Decrements the reference counter, if not of **os::raw_type** (p. 316) or **os::null_type** (p. 316), and delete **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) if needed. Note that if **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) is deleted, so is **os::smart_ptr**<**dataType**>**::ref_count** (p. 389).

Returns

void

12.13.4 Member Data Documentation

template<class dataType> **void_rec os::smart_ptr**< dataType >::func `[private]`

Non-standard deletion.

This is a pointer to a function used when the **os::smart_ptr** (p. 378) is of type **os::shared_type_dynamic_delete** (p. 316).

template<class dataType> dataType* **os::smart_ptr**< dataType >::raw_ptr `[private]`

Pointer to data.

The **os::smart_ptr**<**dataType**>**::raw_ptr** (p. 388) holds the pointer to the block of memory to be managed by the **os::smart_ptr** (p. 378). If this pointer is NULL, the **os::smart_ptr** (p. 378) is of type **os::null_type** (p. 316).

template<class dataType> std::atomic<unsigned long>* **os::smart_ptr**< dataType >::ref_count `[private]`

Reference count.

This pointer stores the current reference count of the **os::smart_ptr** (p. 378). Note that all **os::smart_ptr** (p. 378)'s which point to the same memory address with share the same reference counter. This counter is deleted with the pointer and if this counter is NULL, the **os::smart_ptr** (p. 378) is either of type **os::null_type** (p. 316) or **os::raw_type** (p. 316).

template<class dataType> **smart_pointer_type os::smart_ptr**< dataType >::type `[private]`

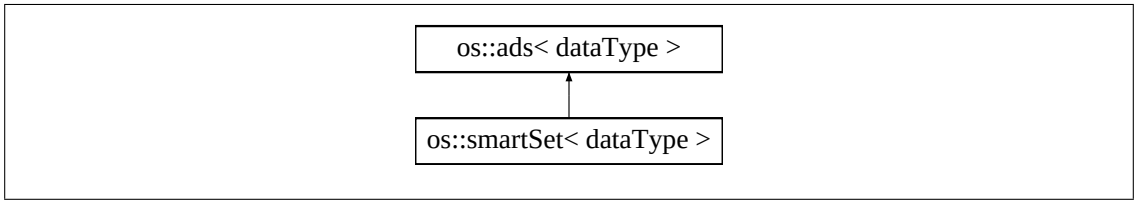
Stores the type.

Defines the type of the **os::smart_ptr** (p. 378). See **os::smart_pointer_type** (p. 316) for details on the available types.

12.14 os::smartSet< dataType > Class Template Reference

Smart set abstract data-structures.

Inheritance diagram for os::smartSet< dataType >:



Public Member Functions

- **smartSet** (**setTypes** typ=**def_set**)

  Default constructor.
- virtual ~**smartSet** ()

  Virtual destructor.
- void **rebuild** (**setTypes** typ)

  Set set type.
- **setTypes getType** () const

  Return set type.
- bool **insert** (**smart_ptr**< **ads**< dataType > > x)

  Inserts an os::ads<dataType>
- bool **insert** (**smart_ptr**< dataType > x)

  Inserts a data node.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< dataType > x)

  Finds a matching node.
- bool **findDelete** (**smart_ptr**< dataType > x)

  Finds and delete a matching node.

- unsigned int **size** () const

  Returns the number of elements in the set.
- **smart_ptr**< **adnode**< dataType > > **getFirst** ()

  Return the first element.
- **smart_ptr**< **adnode**< dataType > > **getLast** ()

  Return the last element.

Private Member Functions

- void **build** (**setTypes** typ)

Private Attributes

- **setTypes type**

  Stores the set type.
- **smart_ptr**< **ads**< dataType > > **current_struct**

  Abstract data-structure storing data.

Additional Inherited Members

12.14.1 Detailed Description

template<class dataType>
class os::smartSet< dataType >

Smart set abstract data-structures.

Wraps other forms of abstract data structures, allowing applications to define abstract data-structures by numbered indexes.

12.14.2 Constructor & Destructor Documentation

template<class dataType > **os::smartSet**< dataType >::**smartSet** (**setTypes** typ = **def_set**) `[inline]`

Default constructor.

This constructor builds the smart set based on a set type. Will call **os::smartSet**<**dataType**>↩ **::build** (p. 391).

Parameters

`in`	*typ*	Set type, default is **os::def_set** (p. 315)

template<class dataType > virtual **os::smartSet**< dataType >::~**smartSet** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.14.3 Member Function Documentation

template<class dataType > void **os::smartSet**< dataType >::build (**setTypes** typ) [inline], [private]

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::smartSet**< dataType >::find (**smart_ptr**< dataType > x) [inline], [virtual]

Finds a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer. Adopts the find function of the abstract data-structure used for this set type. If no abstract data-structure exists, return false.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 326).

template<class dataType > bool **os::smartSet**< dataType >::findDelete (**smart_ptr**< dataType > x) [inline], [virtual]

Finds and delete a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer and remove it. Adopts the findDelete function of the abstract data-structure used for this set type. If no abstract data-structure exists, return false.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::smartSet**< dataType >::getFirst ( ) [inline], [virtual]

Return the first element.

Adopts the getFirst function of the abstract data-structure used for this set type. If no abstract data-structure exists, return NULL.

Returns

os::smartSet<**dataType**>**::current_struct** (p. 393)->**getFirst()** (p. 391)

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::smartSet**< dataType >::getLast ( ) [inline], [virtual]

Return the last element.

Adopts the getLast function of the abstract data-structure used for this set type. If no abstract data-structure exists, return NULL.

Returns

os::smartSet<**dataType**>**::current_struct** (p. 393)->**getLast()** (p. 392)

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **setTypes os::smartSet**< dataType >::getType ( ) const [inline]

Return set type.

Returns

os::smartSet<**dataType**>**::type** (p. 393)

template<class dataType > bool **os::smartSet**< dataType >::insert (**smart_ptr**< **ads**< dataType > > x) [inline], [virtual]

Inserts an os::ads<dataType>

Inserts every element in a given abstract datastructure into this tree. Adopts the insertion function of os::ads<dataType>

[in] x pointer to os::ads<dataType>

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 328).

template<class dataType > bool **os::smartSet**< dataType >::insert (**smart_ptr**< dataType > x) [inline], [virtual]

Inserts a data node.

Adopts the insertion function of the abstract data-structure used for this set type. If no abstract data-structure exists, return false.

[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > void **os::smartSet**< dataType >::rebuild (**setTypes** typ) [inline]

Set set type.

Sets the type of the set, rebuilding the set if the requested type and current type do not match.

Parameters

in	*type*	Set type

Returns

void

template<class dataType > unsigned int **os::smartSet**< dataType >::size ( ) const [inline], [virtual]

Returns the number of elements in the set.

Adopts the size function of the abstract data-structure used for this set type. If no abstract data-structure exists, return 0.

Returns

os::smartSet<**dataType**>**::current_struct** (p. 393)->**size()** (p. 393)

Reimplemented from **os::ads**< **dataType** > (p. 329).

12.14.4 Member Data Documentation

template<class dataType > **smart_ptr**<**ads**<dataType> > **os::smartSet**< dataType >::current_struct [private]

Abstract data-structure storing data.

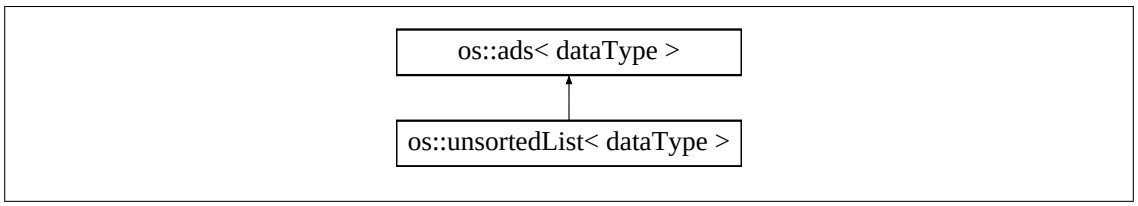
template<class dataType > **setTypes os::smartSet**< dataType >::type [private]

Stores the set type.

12.15 os::unsortedList< dataType > Class Template Reference

Unsorted linked list.

Inheritance diagram for os::unsortedList< dataType >:



Public Member Functions

- **unsortedList** ()

  Default constructor.
- virtual ~**unsortedList** ()

  Virtual destructor.
- bool **insert** (**smart_ptr**< **ads**< dataType > > x)

  Inserts an os::ads<dataType>
- bool **insert** (**smart_ptr**< dataType > x)

  Inserts a data node.
- virtual unsigned int **size** () const

  Returns the number of elements in the list.
- **smart_ptr**< **adnode**< dataType > > **find** (**smart_ptr**< dataType > x)

  Finds a matching node.
- bool **findDelete** (**smart_ptr**< dataType > x)

  Finds and delete a matching node.
- **smart_ptr**< **adnode**< dataType > > **getFirst** ()

  Return the head.
- **smart_ptr**< **adnode**< dataType > > **getLast** ()

  Return the tail.

Private Attributes

- **smart_ptr**< **unsortedListNode**< dataType > > **head**

  Head node.
- **smart_ptr**< **unsortedListNode**< dataType > > **tail**

  Tail node.
- unsigned int **_size**

  Number of elements in the list.

Additional Inherited Members

12.15.1 Detailed Description

template<class dataType>
class os::unsortedList< dataType >

Unsorted linked list.

The list defined by this class is searchable but unsorted. Insert checks to see if the element being inserted is already contained inside the list. Elements are inserted from the front of the list.

12.15.2 Constructor & Destructor Documentation

template<class dataType > **os::unsortedList**< dataType >::**unsortedList** ( ) `[inline]`

Default constructor.

Sets the number of elements to 0 and the head and tail to NULL.

template<class dataType > virtual **os::unsortedList**< dataType >::~**unsortedList** ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called. The list must explicitly force deletion through setting all of the next and previous references of nodes to NULL.

12.15.3 Member Function Documentation

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::unsortedList**< dataType >::find (**smart_ptr**< dataType > x) [inline], [virtual]

Finds a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(n) where n is the number of elements in the list.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 326).

template<class dataType > bool **os::unsortedList**< dataType >::findDelete (**smart_ptr**< dataType > x) [inline], [virtual]

Finds and delete a matching node.

Finds a pointer to an object of type "dataType" given a comparison pointer and removes it. This comparison function is defined by os::adnode<dataType>::compare(smart_ptr<adnode<dataType> >). This function takes O(n) where n is the number of elements in the list.

[in] x dataType pointer to be compared against

Returns

true if the node was found, else false

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::unsortedList**< dataType >::getFirst ( ) [inline], [virtual]

Return the head.

This function is O(1)

Returns

os::unsortedList<**dataType**>**::head** (p. 397)

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::unsortedList**< dataType >::getLast ( ) [inline], [virtual]

Return the tail.
This function is O(1).

Returns

os::unsortedList<**dataType**>**::tail** (p. 397)

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > bool **os::unsortedList**< dataType >::insert (**smart_ptr**< **ads**< dataType > > x) [inline], [virtual]

Inserts an os::ads<dataType>
Inserts every element in a given abstract datastructure into this tree. Adopts the insertion function of os::ads<dataType>
[in] x pointer to os::ads<dataType>

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 328).

template<class dataType > bool **os::unsortedList**< dataType >::insert (**smart_ptr**< dataType > x) [inline], [virtual]

Inserts a data node.
Inserts a pointer to an object of type "dataType." This insertion will place the node into the list at the beginning. If the node already exists, it will not be inserted. This means that this function must first attempt to find the node being inserted. This function is O(n).
[in] x dataType pointer to be inserted

Returns

true if successful, false if failed

Reimplemented from **os::ads**< **dataType** > (p. 327).

template<class dataType > virtual unsigned int **os::unsortedList**< dataType >::size ( ) const [inline], [virtual]

Returns the number of elements in the list.

Returns

os::unsortedList<dataType>::numElements

Reimplemented from **os::ads**< **dataType** > (p. 329).

12.15.4 Member Data Documentation

template<class dataType > unsigned int **os::unsortedList**< dataType >::_size [private]

Number of elements in the list.

template<class dataType > **smart_ptr**<**unsortedListNode**<dataType> > **os::unsortedList**< dataType >::head [private]

Head node.

Contains a pointer to the head node in the list. If this node is NULL, the list is empty.

template<class dataType > **smart_ptr**<**unsortedListNode**<dataType> > **os::unsortedList**< dataType >::tail [private]

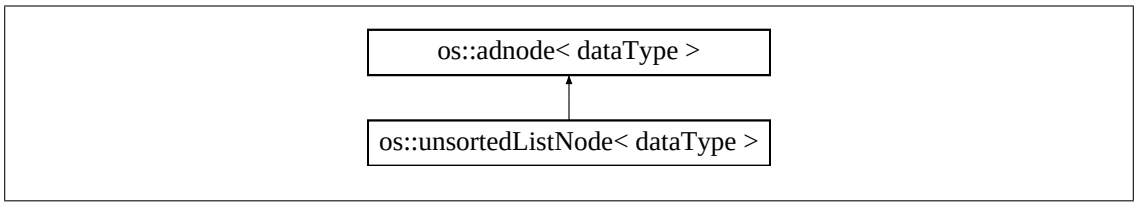
Tail node.

Contains a pointer to the tail node in the list. If this node is NULL, the list is empty.

12.16 os::unsortedListNode< dataType > Class Template Reference

Node for usage in a linked list.

Inheritance diagram for os::unsortedListNode< dataType >:



Public Member Functions

- **unsortedListNode** (**smart_ptr**< dataType > d)

  Abstract data-node constructor.
- virtual ~**unsortedListNode** ()

  Virtual destructor.
- **smart_ptr**< **adnode**< dataType > > **getNext** ()

  Return the next node.
- **smart_ptr**< **adnode**< dataType > > **getPrev** ()

  Return the previous node.

Protected Member Functions

- void **remove** ()

  Remove this node from the list.

Protected Attributes

- **smart_ptr**< **unsortedListNode**< dataType > > **prev**

  Previous node.
- **smart_ptr**< **unsortedListNode**< dataType > > **next**

  Next node.

Friends

- class **unsortedList**< **dataType** >

  List aware of it's nodes.

12.16.1 Detailed Description

template<class dataType>
class os::unsortedListNode< dataType >

Node for usage in a linked list.

This class is a simple extension of the os::adnode<dataType> class. It holds the previous and next node inside of it as well as a pointer to its data. Note that the os::unsortedLIst<dataType> class implements the mechanics of the list.

12.16.2 Constructor & Destructor Documentation

template<class dataType > **os::unsortedListNode**< dataType >::**unsortedListNode** (**smart_ptr**< dataType > d) `[inline]`

Abstract data-node constructor.

A list node is meaningless without a pointer to it's dataType. The constructor requires this pointer to initialize the node. Next and previous nodes are, by default, initialized to zero.

Parameters

`in`	*d*	Data to be bound to the node

template<class dataType > virtual **os::unsortedListNode**< dataType >::~**unsortedListNode** ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.16.3 Member Function Documentation

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::unsortedListNode**< dataType >::getNext ( ) `[inline]`, `[virtual]`

Return the next node.

Note that **os::unsortedListNode**<**dataType**>**::next** (p. 399) is of type os::unsortedListNode<data↩Type>, but this function returns type of os::adnode<dataType>. **os::unsortedListNode**<**data↩Type**>**::next** (p. 399) must be case before returning.

Returns

os::unsortedListNode<**dataType**>**::next** (p. 399)

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType > **smart_ptr**<**adnode**<dataType> > **os::unsortedListNode**< dataType >::getPrev ( ) [inline], [virtual]

Return the previous node.

Note that **os::unsortedListNode**<**dataType**>**::prev** (p. 400) is of type os::unsortedListNode<data↩Type>, but this function returns type of os::adnode<dataType>. **os::unsortedListNode**<**data↩Type**>**::prev** (p. 400) must be case before returning.

Returns

os::unsortedListNode<**dataType**>**::prev** (p. 400)

Reimplemented from **os::adnode**< **dataType** > (p. 324).

template<class dataType > void **os::unsortedListNode**< dataType >::remove ( ) [inline], [protected]

Remove this node from the list.

Removes the references to this node from the next and previous node, if they exists. Sets the previous and next nodes to NULL.

Returns

void

12.16.4 Friends And Related Function Documentation

template<class dataType > friend class **unsortedList**< dataType > [friend]

List aware of it's nodes.

The unsorted list must be aware of the inner-workings of its nodes. Only the unsorted list is permitted to access the private members of this class.

12.16.5 Member Data Documentation

template<class dataType > **smart_ptr**<**unsortedListNode**<dataType> > **os::unsortedListNode**< dataType >::next [protected]

Next node.

Contains a pointer to the next node in the list. If this node is the tail of the list, the next node is NULL.

template<class dataType > **smart_ptr**<**unsortedListNode**<dataType> > **os::unsortedListNode**< dataType >::prev [protected]

Previous node.

Contains a pointer to the previous node in the list. If this node is the head of the list, the previous node is NULL.

12.17 os::vector2d< dataType > Class Template Reference

2-dimensional vector

Public Member Functions

- **vector2d** ()

  Default constructor.
- **vector2d** (dataType xv, dataType yv)

  Value constructor.
- **vector2d** (const **vector2d**< dataType > &vec)

  Copy constructor.
- **vector2d**< dataType > & **operator=** (const **vector2d**< dataType > &vec)

  Equality constructor.
- **vector2d**< dataType > & **operator()** (const dataType &X, const dataType &Y)

  Value setter.
- virtual ~**vector2d** ()

  Virtual destructor s∗ Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.
- dataType **length** () const

  Return length of the vector.
- **vector2d**< dataType > & **scaleSelf** (dataType target=1)

  Scales this vector.
- **vector2d**< dataType > **scale** (dataType target=1) const

  Return a scaled vector.
- int **compare** (const **vector2d**< dataType > &vec) const
- bool **operator==** (const **vector2d**< dataType > &vec) const

  Equality comparison operator.
- bool **operator!=** (const **vector2d**< dataType > &vec) const

  Not-equals comparison operator.
- bool **operator**< (const **vector2d**< dataType > &vec) const

  Less-than comparison operator.
- bool **operator**<**=** (const **vector2d**< dataType > &vec) const

  Less-than or equals to comparison operator.
- bool **operator**> (const **vector2d**< dataType > &vec) const

  Less-than comparison operator.
- bool **operator**>**=** (const **vector2d**< dataType > &vec) const

- **vector2d**< dataType > & **addSelf** (const **vector2d**< dataType > &vec)

  Add vector to self.
- **vector2d**< dataType > **add** (const **vector2d**< dataType > &vec) const

  Add two vectors.
- **vector2d**< dataType > **operator+** (const **vector2d**< dataType > &vec) const

  Add two vectors.
- **vector2d**< dataType > & **operator+=** (const **vector2d**< dataType > &vec)

  Add vector to self.
- **vector2d**< dataType > & **operator++** ()

  Increment.
- **vector2d**< dataType > **operator++** (int dummy)

  Increment.
- **vector2d**< dataType > **operator-** () const

  Invert vector.
- **vector2d**< dataType > & **subtractSelf** (const **vector2d**< dataType > &vec)

  Subtract vector from self.
- **vector2d**< dataType > **subtract** (const **vector2d**< dataType > &vec) const

  Subtract two vectors.
- **vector2d**< dataType > **operator-** (const **vector2d**< dataType > &vec) const

  Subtracts two vectors.
- **vector2d**< dataType > & **operator-=** (const **vector2d**< dataType > &vec)

  Subtracts vector from self.
- **vector2d**< dataType > & **operator--** ()

  Decrement.
- **vector2d**< dataType > **operator--** (int dummy)

  Decrement.
- dataType **dotProduct** (const **vector2d**< dataType > &vec) const

  Dot-product.
- **vector2d**< dataType > **rotate** (const **vector2d**< dataType > &vec) const

  Rotates a point around 0, 0.
- **vector2d**< dataType > **rotateSelf** (const **vector2d**< dataType > &vec)

  Rotates self around 0, 0.

Public Attributes

- dataType **x**

  X axis vector component.
- dataType **y**

  Y axis vector component.

12.17.1 Detailed Description

template<class dataType>
class os::vector2d< dataType >

2-dimensional vector

This template class contains the functions and operators needed to preform arithmetic on a 2 dimensional vector

12.17.2 Constructor & Destructor Documentation

template<class dataType> **os::vector2d**< dataType >::**vector2d** ( ) `[inline]`

Default constructor.

Constructs a 2 dimensional vector with x and y as 0.

template<class dataType> **os::vector2d**< dataType >::**vector2d** (dataType xv, dataType yv) `[inline]`

Value constructor.

Constructs a 2 dimensional vector with a x and a y value.

Parameters

`in`	*xv*	Value of x dimension
`in`	*yv*	Value of y dimension

template<class dataType> **os::vector2d**< dataType >::**vector2d** (const **vector2d**< dataType > & vec) `[inline]`

Copy constructor.

Constructs a 2 dimensional vector from a 2 dimensional vector

Parameters

`in`	*vec*	Vector to be copied

template<class dataType> virtual **os::vector2d**< dataType >::~**vector2d** ( ) `[inline]`, `[virtual]`

Virtual destructor s∗ Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.17.3 Member Function Documentation

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::add (const **vector2d**< dataType > & vec) const [inline]

Add two vectors.

Adds the provided vector to the current vector and returns a new vector. This function is essentially the function version of the '+' operator.

Parameters

in	*vec*	Reference to vector to be added

Returns

Result of the vector addition

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::addSelf (const **vector2d**< dataType > & vec) [inline]

Add vector to self.

Adds the provided vector to the current vector. THis function is essentially the function version of the '+=' operator.

Parameters

in	*vec*	Reference to vector to be added

Returns

Reference to self

template<class dataType> int **os::vector2d**< dataType >::compare (const **vector2d**< dataType > & vec) const [inline]

Compares two vectors

This function compares two vectors for equality. It does not change either vector. This function returns 1 if this object is greater that the object reference received, 0 if the two are equal and -1 if the received reference is greater than the object.

Parameters

in	*vec*	Reference to object compared against

Returns

1 if greater than, 0 if equal to, -1 if less than

template<class dataType> dataType **os::vector2d**< dataType >::dotProduct (const **vector2d**< dataType > & vec) const [inline]

Dot-product.

Calculates the scalar dot-product. Note that this function does not return a vector, but rather, returns a scalar.

Parameters

in	*vec*	Reference to vector

Returns

Scalar dot product

template<class dataType> dataType **os::vector2d**< dataType >::length ( ) const [inline]

Return length of the vector.

Returns sqrt(x^2+y^2), or the length of the vector.

Returns

Length of the vector

template<class dataType> bool **os::vector2d**< dataType >::operator!= (const **vector2d**< dataType > & vec) const [inline]

Not-equals comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if vectors are not equal

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator() (const dataType & X, const dataType & Y) [inline]

Value setter.

Sets the values of a 2 dimensional vector with a x and a y value.

Parameters

in	*X*	Value of x dimension
in	*Y*	Value of y dimension

Returns

Reference to this vector

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::operator+ (const **vector2d**< dataType > & vec) const [inline]

Add two vectors.

Parameters

in	*vec*	Reference to vector to be added

Returns

vector2d<dataType>::add(vec)

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator++ ( ) [inline]

Increment.

Increments this vector by the unit vector of the same direction and then returns a reference to this vector.

Returns

Reference to self

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::operator++ (int dummy) [inline]

Increment.

Copies this vector then increments this vector by the unit vector of the same direction and then returns the original copy.

Parameters

in	*dummy*	Parameter required to define operator

Returns

Original copy

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator+= (const **vector2d**< dataType > & vec) [inline]

Add vector to self.

Parameters

in	*vec*	Reference to vector to be added

Returns

vector3d<dataType>::addSelf(vec)

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::operator- ( ) const `[inline]`

Invert vector.

Constructs a new vector with an inverted x and inverted y.

Returns

Inverted vector

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::operator- (const **vector2d**< dataType > & vec) const `[inline]`

Subtracts two vectors.

Parameters

`in`	*vec*	Reference to vector to be subtracted

Returns

vector2d<dataType>::subtract(vec)

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator-- ( ) `[inline]`

Decrement.

Decrements this vector by the unit vector of the same direction and then returns a reference to this vector.

Returns

Reference to self

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::operator-- (int dummy) `[inline]`

Decrement.

Copies this vector then decrements this vector by the unit vector of the same direction and then returns the original copy.

Parameters

`in`	*dummy*	Parameter required to define operator

Returns

Original copy

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator-= (const **vector2d**< dataType > & vec) [inline]

Subtracts vector from self.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

vector3d<dataType>::subtractSelf(vec)

template<class dataType> bool **os::vector2d**< dataType >::operator< (const **vector2d**< dataType > & vec) const [inline]

Less-than comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is less than vec

template<class dataType> bool **os::vector2d**< dataType >::operator<= (const **vector2d**< dataType > & vec) const [inline]

Less-than or equals to comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is less than vec

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::operator= (const **vector2d**< dataType > & vec) [inline]

Equality constructor.
Set the values of a 2 dimensional vector from a another 2 dimensional vector

Parameters

in	*vec*	Vector to be copied

Returns

Reference to this vector

template<class dataType> bool **os::vector2d**< dataType >::operator== (const **vector2d**< dataType > & vec) const [inline]

Equality comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if vectors are equal

template<class dataType> bool **os::vector2d**< dataType >::operator> (const **vector2d**< dataType > & vec) const [inline]

Less-than comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is less than vec

template<class dataType> bool **os::vector2d**< dataType >::operator>= (const **vector2d**< dataType > & vec) const [inline]

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::rotate (const **vector2d**< dataType > & vec) const [inline]

Rotates a point around 0, 0.

Parameters

in	*vec*	Vector representing an angle

Returns

Rotated point

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::rotateSelf (const **vector2d**< dataType > & vec) [inline]

Rotates self around 0, 0.

Parameters

in	*vec*	Vector representing an angle

Returns

Rotated point

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::scale (dataType target = 1) const [inline]

Return a scaled vector.

Returns a vector scaled to the given target length. This operation, by default, will scale to a distance of 1 (the unit vector)

Parameters

in	*target*	Vector length to be scaled to

Returns

The scaled vector

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::scaleSelf (dataType target = 1) [inline]

Scales this vector.

Scales this vector to the given target length. This operation, by default, will scale to a distance of 1 (the unit vector)

Parameters

in	*target*	Vector length to be scaled to

Returns

Reference to this

template<class dataType> **vector2d**<dataType> **os::vector2d**< dataType >::subtract (const **vector2d**< dataType > & vec) const [inline]

Subtract two vectors.

Subtracts the provided vector from the current vector and returns a new vector. This function is essentially the function version of the '-' operator.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

Result of the vector subtraction

template<class dataType> **vector2d**<dataType>& **os::vector2d**< dataType >::subtractSelf (const **vector2d**< dataType > & vec) [inline]

Subtract vector from self.

Subtracts the provided vector from the current vector. THis function is essentially the function version of the '-=' operator.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

Reference to self

12.17.4 Member Data Documentation

template<class dataType> dataType **os::vector2d**< dataType >::x

X axis vector component.

template<class dataType> dataType **os::vector2d**< dataType >::y

Y axis vector component.

12.18 os::vector3d< dataType > Class Template Reference

3-dimensional vector

Public Member Functions

- **vector3d** ()

  Default constructor.
- **vector3d** (dataType xv, dataType yv, dataType zv=0)

  Value constructor.
- **vector3d** (const **vector3d**< dataType > &vec)

  Copy constructor.
- **vector3d** (const **vector2d**< dataType > &vec)

  Copy constructor.
- **vector3d**< dataType > & **operator=** (const **vector3d**< dataType > &vec)

  Equality constructor.
- **vector3d**< dataType > & **operator()** (const dataType &X, const dataType &Y, const dataType &Z)

  Value setter.
- virtual ~**vector3d** ()

  Virtual destructor s∗ Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.
- dataType **length** () const

  Return length of the vector.
- **vector3d**< dataType > & **scaleSelf** (dataType target=1)

  Scales this vector.
- **vector3d**< dataType > **scale** (dataType target=1) const

  Return a scaled vector.
- int **compare** (const **vector3d** &vec) const
- bool **operator==** (const **vector3d**< dataType > &vec) const

  Equality comparison operator.
- bool **operator!=** (const **vector3d**< dataType > &vec) const

  Not-equals comparison operator.
- bool **operator**< (const **vector3d**< dataType > &vec) const

  Less-than comparison operator.
- bool **operator**<**=** (const **vector3d**< dataType > &vec) const

  Less-than or equal to comparison operator.
- bool **operator**> (const **vector3d**< dataType > &vec) const

  Greater-than comparison operator.
- bool **operator**>**=** (const **vector3d**< dataType > &vec) const

  Greater-than or equal to comparison operator.
- **vector3d**< dataType > & **addSelf** (const **vector3d**< dataType > &vec)

  Add vector to self.
- **vector3d**< dataType > **add** (const **vector3d**< dataType > &vec) const

  Add two vectors.
- **vector3d**< dataType > **operator+** (const **vector3d**< dataType > &vec) const

  Add two vectors.
- **vector3d**< dataType > & **operator+=** (const **vector3d**< dataType > &vec)

Add vector to self.

- **vector3d**< dataType > & **operator++** ()

  Increment.
- **vector3d**< dataType > **operator++** (int dummy)

  Increment.
- **vector3d**< dataType > **operator-** () const

  Invert vector.
- **vector3d**< dataType > & **subtractSelf** (const **vector3d**< dataType > &vec)

  Subtract vector from self.
- **vector3d**< dataType > **subtract** (const **vector3d**< dataType > &vec) const

  Subtract two vectors.
- **vector3d**< dataType > **operator-** (const **vector3d**< dataType > &vec) const

  Subtracts two vectors.
- **vector3d**< dataType > & **operator-=** (const **vector3d**< dataType > &vec)

  Subtracts vector from self.
- **vector3d**< dataType > & **operator--** ()

  Decrement.
- **vector3d**< dataType > **operator--** (int dummy)

  Decrement.
- dataType **dotProduct** (const **vector3d**< dataType > &vec) const

  Dot-product.
- **vector3d**< dataType > **crossProduct** (const **vector3d**< dataType > &vec) const

  Cross-product.
- **vector3d**< dataType > & **crossSelf** (const **vector3d**< dataType > &vec)

  Cross-product to self.
- **vector3d**< dataType > **operator**∗ (const **vector3d**< dataType > &vec) const

  Cross-product.
- **vector3d**< dataType > & **operator**∗**=** (const **vector3d**< dataType > &vec)

  Self cross-product.

Public Attributes

- dataType **x**

  X axis vector component.
- dataType **y**

  Y axis vector component.
- dataType **z**

  Z axis vector component.

12.18.1 Detailed Description

template<class dataType>
class os::vector3d< dataType >

3-dimensional vector

This template class contains the functions and operators needed to preform arithmetic on a 3 dimensional vector

12.18.2 Constructor & Destructor Documentation

template<class dataType> **os::vector3d**< dataType >::**vector3d** ( ) `[inline]`

Default constructor.

Constructs a 3 dimensional vector with x, y and z as 0.

template<class dataType> **os::vector3d**< dataType >::**vector3d** (dataType xv, dataType yv, dataType zv = `0`) `[inline]`

Value constructor.

Constructs a 3 dimensional vector with x, y and z values. Z, by default, is initialized as 0.

Parameters

`in`	*xv*	Value of x dimension
`in`	*yv*	Value of y dimension
`in`	*zv*	Value of z dimension

template<class dataType> **os::vector3d**< dataType >::**vector3d** (const **vector3d**< dataType > & vec) `[inline]`

Copy constructor.

Constructs a 3 dimensional vector from another 3 dimensional vector

Parameters

`in`	*vec*	Vector to be copied

Returns

Reference to this vector

template<class dataType> **os::vector3d**< dataType >::**vector3d** (const **vector2d**< dataType > & vec) `[inline]`

Copy constructor.

Constructs a 3 dimensional vector from a 2 dimensional vector

Parameters

in	*vec*	Vector to be copied

Returns

Reference to this vector

template<class dataType> virtual **os::vector3d**< dataType >::~**vector3d** ( ) [inline], [virtual]

Virtual destructor s∗ Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

12.18.3 Member Function Documentation

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::add (const **vector3d**< dataType > & vec) const [inline]

Add two vectors.

Adds the provided vector to the current vector and returns a new vector. This function is essentially the function version of the '+' operator.

Parameters

in	*vec*	Reference to vector to be added

Returns

Result of the vector addition

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::addSelf (const **vector3d**< dataType > & vec) [inline]

Add vector to self.

Adds the provided vector to the current vector. THis function is essentially the function version of the '+=' operator.

Parameters

in	*vec*	Reference to vector to be added

Returns

Reference to self

template<class dataType> int **os::vector3d**< dataType >::compare (const **vector3d**< dataType > & vec) const [inline]

Compares two vectors

This function compares two vectors for equality. It does not change either vector. This function returns 1 if this object is greater that the object reference received, 0 if the two are equal and -1 if the received reference is greater than the object.

Parameters

in	*vec*	Reference to object compared against

Returns

1 if greater than, 0 if equal to, -1 if less than

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::crossProduct (const **vector3d**< dataType > & vec) const [inline]

Cross-product.

Preform the cross-product computation on this vector and the vector argument provided. Unlike the dot-product, the cross product returns a vector.

Parameters

in	*vec*	Reference to vector to be computed

Returns

Result of the cross-product

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::crossSelf (const **vector3d**< dataType > & vec) [inline]

Cross-product to self.

Preform the cross-product computation on this vector and the vector argument provided. Binds the result to this and returns a reference to this vector.

Parameters

in	*vec*	Reference to vector to be computed

Returns

Reference to self

template<class dataType> dataType **os::vector3d**< dataType >::dotProduct (const **vector3d**< dataType > & vec) const [inline]

Dot-product.

Calculates the scalar dot-product. Note that this function does not return a vector, but rather, returns a scalar.

Parameters

in	*vec*	Reference to vector

Returns

Scalar dot product

template<class dataType> dataType **os::vector3d**< dataType >::length ( ) const [inline]

Return length of the vector.

Returns sqrt(x^2+y^2+z^2), or the length of the vector.

Returns

Length of the vector

template<class dataType> bool **os::vector3d**< dataType >::operator!= (const **vector3d**< dataType > & vec) const [inline]

Not-equals comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if vectors are not equal

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator() (const dataType & X, const dataType & Y, const dataType & Z) [inline]

Value setter.

Sets values of a 3 dimensional vector with x, y and z values.

Parameters

in	*X*	Value of x dimension

Parameters

in	*Y*	Value of y dimension
in	*Z*	Value of z dimension

Returns

Reference to this vector

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator∗ (const **vector3d**< dataType > & vec) const [inline]

Cross-product.

Parameters

in	*vec*	Reference to vector to be computed with

Returns

vector3d<dataType>::crossProduct(vec)

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator∗= (const **vector3d**< dataType > & vec) [inline]

Self cross-product.

Parameters

in	*vec*	Reference to vector to be computed with

Returns

vector3d<dataType>::crossSelf(vec)

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator+ (const **vector3d**< dataType > & vec) const [inline]

Add two vectors.

Parameters

in	*vec*	Reference to vector to be added

Returns

vector3d<dataType>::add(vec)

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator++ ( ) [inline]

Increment.

Increments this vector by the unit vector of the same direction and then returns a reference to this vector.

Returns

Reference to self

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator++ (int dummy) [inline]

Increment.

Copies this vector then increments this vector by the unit vector of the same direction and then returns the original copy.

Parameters

in	*dummy*	Parameter required to define operator

Returns

Original copy

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator+= (const **vector3d**< dataType > & vec) [inline]

Add vector to self.

Parameters

in	*vec*	Reference to vector to be added

Returns

vector3d<dataType>::addSelf(vec)

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator- ( ) const [inline]

Invert vector.

Constructs a new vector with an inverted x, inverted y and inverted z.

Returns

Inverted vector

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator- (const **vector3d**< dataType > & vec) const [inline]

Subtracts two vectors.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

vector3d<dataType>::subtract(vec)

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator-- ( ) [inline]

Decrement.

Decrements this vector by the unit vector of the same direction and then returns a reference to this vector.

Returns

Reference to self

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::operator-- (int dummy) [inline]

Decrement.

Copies this vector then decrements this vector by the unit vector of the same direction and then returns the original copy.

Parameters

in	*dummy*	Parameter required to define operator

Returns

Original copy

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator-= (const **vector3d**< dataType > & vec) [inline]

Subtracts vector from self.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

vector3d<dataType>::subtractSelf(vec)

template<class dataType> bool **os::vector3d**< dataType >::operator< (const **vector3d**< dataType > & vec) const [inline]

Less-than comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is less than vec

template<class dataType> bool **os::vector3d**< dataType >::operator<= (const **vector3d**< dataType > & vec) const [inline]

Less-than or equal to comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is less than or equal to vec

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::operator= (const **vector3d**< dataType > & vec) [inline]

Equality constructor.
Set the values of a 3 dimensional vector from a another 3 dimensional vector

Parameters

in	*vec*	Vector to be copied

Returns

Reference to this vector

template<class dataType> bool **os::vector3d**< dataType >::operator== (const **vector3d**< dataType > & vec) const [inline]

Equality comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if vectors are equal

template<class dataType> bool **os::vector3d**< dataType >::operator> (const **vector3d**< dataType > & vec) const [inline]

Greater-than comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is greater than vec

template<class dataType> bool **os::vector3d**< dataType >::operator>= (const **vector3d**< dataType > & vec) const [inline]

Greater-than or equal to comparison operator.

Parameters

in	*vec*	Reference to object compared against

Returns

true if this is greater than or equal to vec

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::scale (dataType target = 1) const [inline]

Return a scaled vector.

Returns a vector scaled to the given target length. This operation, by default, will scale to a distance of 1 (the unit vector)

Parameters

in	*target*	Vector length to be scaled to

Returns

The scaled vector

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::scaleSelf (dataType target = 1) [inline]

Scales this vector.

Scales this vector to the given target length. This operation, by default, will scale to a distance of 1 (the unit vector)

Parameters

in	*target*	Vector length to be scaled to

Returns

Reference to this

template<class dataType> **vector3d**<dataType> **os::vector3d**< dataType >::subtract (const **vector3d**< dataType > & vec) const [inline]

Subtract two vectors.

Subtracts the provided vector to the current vector and returns a new vector. This function is essentially the function version of the '-' operator.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

Result of the vector subtraction

template<class dataType> **vector3d**<dataType>& **os::vector3d**< dataType >::subtractSelf (const **vector3d**< dataType > & vec) [inline]

Subtract vector from self.

Subtracts the provided vector from the current vector. THis function is essentially the function version of the '-=' operator.

Parameters

in	*vec*	Reference to vector to be subtracted

Returns

Reference to self

12.18.4 Member Data Documentation

template<class dataType> dataType **os::vector3d**< dataType >::x

X axis vector component.

template<class dataType> dataType **os::vector3d**< dataType >::y

Y axis vector component.

template<class dataType> dataType **os::vector3d**< dataType >::z

Z axis vector component.

Part III

Unit Test Library

Chapter 13

Introduction

The UnitTest library contains classes which preform automated unit tests while a project is under development. Utilizing C++ exceptions, the UnitTest library separates its test battery into libraries tested, suites in libraries and tests in suites. The UnitTest library iterates through instantiated libraries running every test suite in the library.

13.1 Namespace test

The test namespace is designed to hold all of the classes and functions related to unit testing. Classes and functions in the test namespace should not be included in the final release application. It is expected that libraries add to this namespace and place their own testing assets here. Note that the test namespace uses elements from the os namespace, all of these elements are defined in the Datastructures library.

13.2 Datastructures Testing

The Datastructures library is rigorously unit tested by the UnitTest library, and the Datastructures unit tests are automatically included in any system unit test unless specifically removed. The Datastructures UnitTests are particularly important because the Datastructures library serves as a base for memory management and data organization. These tests fall broadly into two categories: deterministic and random.

Deterministic tests preform the exact same test every iteration. Deterministic tests are used to ensure that specific functions and operators are returning expected data. Deterministic tests don't merely identify the existence of an error, but usually identify the precise nature of the error as well.

Random tests use a random number generator to preform a unique test with every iteration. This allows unit tests to, over time, catch edge cases with complex data structures. In contrast to deterministic tests, random testing will usually not identify the precise nature of the error.

Note that as a general rule, the implementation of tests is not documented. The location of test suites is documented, through both .h and .cpp files, but the classes and functions which make up these tests are not included.

Chapter 14

File Index

14.1 File List

Here is a list of all files with brief descriptions:

Chapter 15

File Documentation

15.1 DatastructuresTest.h File Reference

Datastructures library test.

15.1.1 Detailed Description

Datastructures library test.

Author

Jonathan Bedard

Date

2/4/2016

Bug No known bugs.

Contains the declaration of the Datastructures library test. Note that this library test is automatically added to all Unit Test executables.

15.2 DatastructuresTest.cpp File Reference

Datastructures library test implementation.

15.2.1 Detailed Description

Datastructures library test implementation.

Author

Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

Implements the Datastructures library test. These tests are designed to guarantee the functionality of each of the elements in the Datastructures library.

15.3 masterTestHolder.h File Reference

Library tests, masterTestHolder singleton.

Classes

- class **test::libraryTests**

  Library test group.
- class **test::masterTestHolder**

  Unit Test singleton.

Namespaces

- **test**

15.3.1 Detailed Description

Library tests, masterTestHolder singleton.
Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains declarations for the library test base class and **test::masterTestHolder** (p. 447) singleton class. This file represents the top level of the Unit Test driver classes.

15.4 masterTestHolder.cpp File Reference

Library tests, masterTestHolder singleton implementations.

15.4.1 Detailed Description

Library tests, masterTestHolder singleton implementations.
Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains implementations for the library test base class and **test::masterTestHolder** (p. 447) singleton class. Consult **masterTestHolder.h** (p. 428) for details.

15.5 singleTest.h File Reference

Single test class.

Classes

- class **test::singleTest**

  Single unit test class.
- class **test::singleFunctionTest**

  Single unit test from function.

Namespaces

- **test**

Typedefs

- typedef void(∗ **test::testFunction**) ()

  Typedef for single test function.

15.5.1 Detailed Description

Single test class.
Jonathan Bedard

Date

2/6/2016

Bug No known bugs.

This file contains declarations for a single unit test. Unit tests can be defined as separate class or a simple test function.

15.6 singleTest.cpp File Reference

Single test class implementation.

15.6.1 Detailed Description

Single test class implementation.
Jonathan Bedard

Date

2/6/2016

Bug No known bugs.

This file contains implementation for a single unit test. Consult singeTest.h for details.

15.7 TestSuite.h File Reference

Single test class.

Classes

- class **test::testSuite**

Namespaces

- **test**

15.7.1 Detailed Description

Single test class.
Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This file contains declarations for a test suite. Test suites contain lists of unit tests.

15.8 TestSuite.cpp File Reference

Single test class.

15.8.1 Detailed Description

Single test class.
Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This file contains declarations for a test suite. Consult **testSuite.h** (p. 430) for details.

15.9 UnitTest.h File Reference

Unit Test header file.

Namespaces

- **test**

Functions

- void **test::startTests** ()

  Print out header for Unit Tests.
- void **test::endTestsError** (os::smart_ptr< std::exception > except)

  End tests in error.
- void **test::endTestsSuccess** ()

  End tests successfully.
- void **test::testInit** (int argc=0, char **argv=NULL)

  Test initialization.

15.9.1 Detailed Description

Unit Test header file.

Author

Jonathan Bedard

Date

4/2/2016

Bug No known bugs.

Packages all headers required for the UnitTest library and declares a number of global test functions used for initializing and ending a Unit Test battery.

15.10 UnitTest.cpp File Reference

Unit Test logging and global functions.

15.10.1 Detailed Description

Unit Test logging and global functions.

Author

Jonathan Bedard

Date

2/4/2016

Bug No known bugs.

Implements logging in the test namespace. Implements a number of global test functions used for initializing and ending a Unit Test battery.

15.11 UnitTestLog.h File Reference

Namespaces

- **test**

Functions

- std::ostream & **test::testout_func** ()

  Standard out object for test namespace.
- std::ostream & **test::testerr_func** ()

  Standard error object for test namespace.

Variables

- os::smart_ptr< std::ostream > **test::testout_ptr**

  Standard out pointer for test namespace.
- os::smart_ptr< std::ostream > **test::testerr_ptr**

  Standard error pointer for test namespace.

15.12 UnitTestExceptions.h File Reference

Common exceptions thrown by unit tests.

Classes

- class **test::generalTestException**

  Base class for test exceptions.
- class **test::unknownException**

  Unknown exception class.
- class **test::nullFunctionException**

  NULL function exception class.

Namespaces

- **test**

15.12.1 Detailed Description

Common exceptions thrown by unit tests.

Jonathan Bedard

Date

2/19/2016

Bug No known bugs.

This file contains a number of common test exceptions used by unit tests. All of these classes extend std::exception.

Chapter 16

Class Index

16.1 Class List

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

Chapter 17

Namespace Documentation

17.1 test Namespace Reference

Classes

- class **generalTestException**
  Base class for test exceptions.
- class **libraryTests**
  Library test group.
- class **masterTestHolder**
  Unit Test singleton.
- class **nullFunctionException**
  NULL function exception class.
- class **singleFunctionTest**
  Single unit test from function.
- class **singleTest**
  Single unit test class.
- class **testSuite**
- class **unknownException**
  Unknown exception class.

Typedefs

- typedef void(∗ **testFunction**) ()
  Typedef for single test function.

Functions

- void **startTests** ()
  Print out header for Unit Tests.
- void **endTestsError** (os::smart_ptr< std::exception > except)
  End tests in error.

- void **endTestsSuccess** ()

  End tests successfully.
- void **testInit** (int argc=0, char ∗∗argv=NULL)

  Test initialization.
- std::ostream & **testout_func** ()

  Standard out object for test namespace.
- std::ostream & **testerr_func** ()

  Standard error object for test namespace.

Variables

- os::smart_ptr< std::ostream > **testout_ptr**

  Standard out pointer for test namespace.
- os::smart_ptr< std::ostream > **testerr_ptr**

  Standard error pointer for test namespace.

17.1.1 Typedef Documentation

typedef void(∗ test::testFunction) ()

Typedef for single test function.

This typedef defines what a single test function looks like. For simplicity, a single unit test can be defined by a function of this type instead of inheriting from **test::singleTest** (p. 452).

Returns

void

17.1.2 Function Documentation

void test::endTestsError (os::smart_ptr< std::exception > except)

End tests in error.

Prints out a global division block line of '=' characters, then the information provided in the exception passed to the function then another global division block

Parameters

in	*except*	Exception which caused the error

Returns

void

void test::endTestsSuccess ( )

End tests successfully.

Prints out a global division block line of '=' characters, then the test results data provided by the **test::masterTestHolder** (p. 447) then another global division block

Returns

void

void test::startTests ( )

Print out header for Unit Tests.

Prints out a global division block line of '=' characters, then 'Unit Test Battery' and then another global division block.

Returns

void

std::ostream& test::testerr_func ( )

Standard error object for test namespace.

#define statements allow the user to call this function with "test::testerr." Logging is achieved by using "test::testerr" as one would use "std::cerr."

void test::testInit (int argc = 0, char ** argv = NULL)

Test initialization.

This function is re-implemented by each executable which uses the UnitTest library. This function is used to bind all of the library tests, except the Datastructures library test.

Returns

void

std::ostream& test::testout_func ( )

Standard out object for test namespace.

#define statements allow the user to call this function with "test::testout." Logging is achieved by using "test::testout" as one would use "std::cout."

17.1.3 Variable Documentation

os::smart_ptr<std::ostream> test::testerr_ptr

Standard error pointer for test namespace.

This std::ostream is used as standard error for the test namespace. This pointer can be swapped out to programmatically redirect standard error for the test namespace.

os::smart_ptr<std::ostream> test::testout_ptr

Standard out pointer for test namespace.

This std::ostream is used as standard out for the test namespace. This pointer can be swapped out to programmatically redirect standard out for the test namespace.

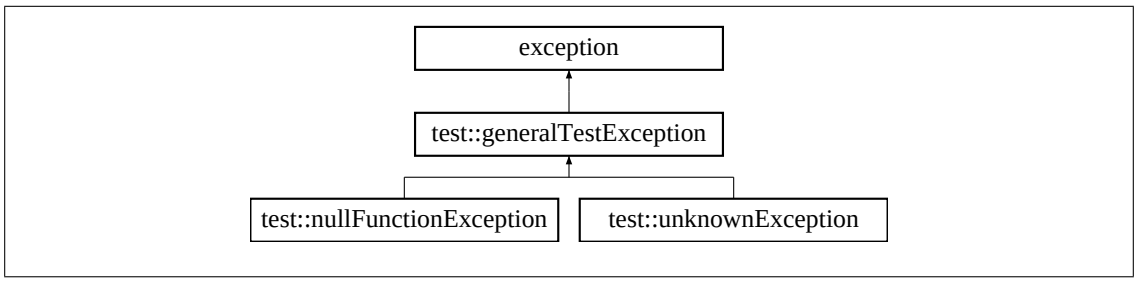
Chapter 18

Class Documentation

18.1 test::generalTestException Class Reference

Base class for test exceptions.

Inheritance diagram for test::generalTestException:



Public Member Functions

- **generalTestException** (std::string err, std::string loc)

  Construct exception with error and location.
- virtual ~**generalTestException** () throw ()

  Virtual destructor.
- virtual const char * **what** () const throw ()

  std::exception overload
- const std::string & **getLocation** () const

  Location description.
- const std::string & **getString** () const

  Error description.

Private Attributes

- std::string **location**

  The location where the error came from.

- std::string **_error**

  A description of the error.
- std::string **total_error**

  Combination of the error and location.

18.1.1 Detailed Description

Base class for test exceptions.

This class defines an exception which has a location. Because this class holds multiple std::string objects, the error description can be dynamically set.

18.1.2 Constructor & Destructor Documentation

test::generalTestException::generalTestException (std::string err, std::string loc) `[inline]`

Construct exception with error and location.

Constructs the exception with an error string and a location string. Also builds the **test::general↩ TestException::total_error** (p. 440) string for use by the "what()" function.

Parameters

`in`	*err*	Error string
`in`	*loc*	Location string

virtual test::generalTestException::~generalTestException ( ) throw) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.1.3 Member Function Documentation

const std::string& test::generalTestException::getLocation ( ) const `[inline]`

Location description.

Returns

test::generalTestException::location (p. 440)

const std::string& test::generalTestException::getString ( ) const `[inline]`

Error description.

Returns

test::generalTestException::_error (p. 440)

virtual const char* test::generalTestException::what ( ) const throw) [inline], [virtual]

std::exception overload

Overloaded from std::exception. This function outputs the complete description, which contains both the error description and location description.

Returns

character pointer to the complete description

18.1.4 Member Data Documentation

std::string test::generalTestException::_error [private]

A description of the error.

std::string test::generalTestException::location [private]

The location where the error came from.

std::string test::generalTestException::total_error [private]

Combination of the error and location.

This string is constructed in the constructor so that "what()" can refer to a location in memory. This std::string is a combination of **test::generalTestException::_error** (p. 440) and **test::generalTest↩ Exception::location** (p. 440).

18.2 test::libraryTests Class Reference

Library test group.

Public Member Functions

- **libraryTests** (std::string ln)

  Library test constructor.
- virtual ~**libraryTests** ()

  Virtual destructor.
- void **runTests** () throw (os::smart_ptr<std::exception>)

  Runs all of the test suites.
- virtual void **onSetup** ()

  Runs on shutdown of the group.
- virtual void **onTeardown** ()

  Runs on teardown of the group.
- void **logBegin** ()

  Logs the beginning of a library test.
- bool **logEnd** (os::smart_ptr< std::exception > except=NULL)

  Logs the end of a library test.

- int **getNumSuites** () const

  Number of suites in the set.
- int **getNumSuccess** () const

  Number of suites successfully completed.
- int **getNumRun** () const

  Number of suites attempted to run.
- void **pushSuite** (os::smart_ptr< **testSuite** > suite)

  Add suite to the set.
- void **removeSuite** (os::smart_ptr< **testSuite** > suite)

  Remove suite from the set.
- bool **operator==** (const **libraryTests** &lt) const

  Equality comparison.
- bool **operator!=** (const **libraryTests** &lt) const

  Not-equals comparison.
- bool **operator**> (const **libraryTests** &lt) const

  Greater-than comparison.
- bool **operator**< (const **libraryTests** &lt) const

  Less-than comparison.
- bool **operator**>**=** (const **libraryTests** &lt) const

  Greater-than or equal to comparison.
- bool **operator**<**=** (const **libraryTests** &lt) const

  Less-than or equal to comparison.

Private Attributes

- std::string **libName**

  Name of library to be tested.
- os::smartSet< **testSuite** > **suiteList**

  Set of test suites.
- int **suitesCompleted**

  Number of suites successfully completed.
- int **suitesRun**

  Number of suites attempted to run.

18.2.1 Detailed Description

Library test group.

This class contains a set of test suites which are designed to a specific library. Each library must define it's own version of this class in-order to be tested.

18.2.2 Constructor & Destructor Documentation

test::libraryTests::libraryTests (std::string ln)

Library test constructor.

This constructor initializes the number of suites completed and number of suites run to 0, along with sets the name of library being tested.

Parameters

in	*ln*	Name of library to be tested

virtual test::libraryTests::~libraryTests ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.2.3 Member Function Documentation

int test::libraryTests::getNumRun ( ) const [inline]

Number of suites attempted to run.

Returns

test::libraryTests::suitesRun (p. 447)

int test::libraryTests::getNumSuccess ( ) const [inline]

Number of suites successfully completed.

Returns

test::libraryTests::suitesCompleted (p. 447)

int test::libraryTests::getNumSuites ( ) const [inline]

Number of suites in the set.

Returns

test::libraryTests::suiteList.size()

void test::libraryTests::logBegin ( )

Logs the beginning of a library test.

Outputs the name of the library to be tested along with a line break made of '+' characters.

Returns

void

bool test::libraryTests::logEnd (os::smart_ptr< std::exception > except = NULL)

Logs the end of a library test.

Outputs the number of suites run and how many of these suites were both successful and how many of these suites failed.

Returns

True if all suites successful, else false

virtual void test::libraryTests::onSetup ( ) [inline], [virtual]

Runs on shutdown of the group.

Each library group calls this funciton as it starts up, allowing groups to define actions performed to setup the group.

Returns

void

virtual void test::libraryTests::onTeardown ( ) [inline], [virtual]

Runs on teardown of the group.

Guranteed to run even if the group itself fails. A custom tear-down for the group can re-impliment this class.

Returns

void

bool test::libraryTests::operator!= (const **libraryTests** & lt) const [inline]

Not-equals comparison.

Compares two test::libraryTest based on the library name. If the two names are not-equal, the library tests are not-equal.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName!=lt.libName

bool test::libraryTests::operator< (const **libraryTests** & lt) const [inline]

Less-than comparison.

Compares two test::libraryTest based on the library name. If the name of this object is less than the name of the reference object, return true.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName<lt.libName

bool test::libraryTests::operator<= (const **libraryTests** & lt) const [inline]

Less-than or equal to comparison.

Compares two test::libraryTest based on the library name. If the name of this object is less than or equal to the name of the reference object, return true.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName<=lt.libName

bool test::libraryTests::operator== (const **libraryTests** & lt) const [inline]

Equality comparison.

Compares two test::libraryTest based on the library name. If the two names are equal, the library tests are equal.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName==lt.libName

bool test::libraryTests::operator> (const **libraryTests** & lt) const [inline]

Greater-than comparison.

Compares two test::libraryTest based on the library name. If the name of this object is greater than the name of the reference object, return true.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName>lt.libName

bool test::libraryTests::operator>= (const **libraryTests** & lt) const [inline]

Greater-than or equal to comparison.

Compares two test::libraryTest based on the library name. If the name of this object is greater than or equal to the name of the reference object, return true.

Parameters

in	*lt*	Reference to test::libraryTest to be compared against

Returns

this->libName>=lt.libName

void test::libraryTests::pushSuite (os::smart_ptr< **testSuite** > suite) [inline]

Add suite to the set.

Adds a **test::testSuite** (p. 455) to the set of suites to be tested.

Parameters

in	*suite*	Test suite to be added to set

Returns

void

void test::libraryTests::removeSuite (os::smart_ptr< **testSuite** > suite) [inline]

Remove suite from the set.

Removes a **test::testSuite** (p. 455) from the set of suites to be tested.

Parameters

in	*suite*	Test suite to be removed from the set

Returns

void

void test::libraryTests::runTests ( ) throw os::smart_ptr< std::exception >)

Runs all of the test suites.

Runs all test suites bound to this class. Each suite should manage its own errors, but it is possible that this function will throw an error of type os::smart_ptr<std::exception>.

Returns

void

18.2.4 Member Data Documentation

std::string test::libraryTests::libName [private]

Name of library to be tested.

os::smartSet<**testSuite**> test::libraryTests::suiteList [private]

Set of test suites.

int test::libraryTests::suitesCompleted [private]

Number of suites successfully completed.

int test::libraryTests::suitesRun [private]

Number of suites attempted to run.

18.3 test::masterTestHolder Class Reference

Unit Test singleton.

Public Member Functions

- virtual ~**masterTestHolder** ()
  Virtual destructor.
- bool **runTests** () throw (os::smart_ptr<std::exception>)
  Runs all of the library tests.
- int **getNumLibs** () const
  Number of libraries in the set.
- int **getNumSuccess** () const
  Number of libraries successfully completed.
- int **getNumRun** () const
  Number of libraries attempted to run.
- void **pushLibrary** (os::smart_ptr< **libraryTests** > lib)
  Add library to the set.
- void **removeLibrary** (os::smart_ptr< **libraryTests** > lib)
  Remove library from the set.

Static Public Member Functions

- static os::smart_ptr< **masterTestHolder** > **singleton** ()
  Singleton access.

Private Member Functions

- **masterTestHolder** ()
  Private constructor.

Private Attributes

- os::smartSet< **libraryTests** > **libraryList**

  Set of library tests.
- int **libsCompleted**

  Number of libraries successfully completed.
- int **libsRun**

  Number of libraries attempted to run.

18.3.1 Detailed Description

Unit Test singleton.

This class contains a set of library tests. Every library test must add itself to this class in-order to be tested. The **test::masterTestHolder::runTests()** (p. 449) function runs all of the library tests.

18.3.2 Constructor & Destructor Documentation

test::masterTestHolder::masterTestHolder ( ) `[private]`

Private constructor.

The **test::masterTestHolder** (p. 447) class is a singleton class. This constructor initializes the number of libraries completed and number of libraries run to 0.

virtual test::masterTestHolder::~masterTestHolder ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.3.3 Member Function Documentation

int test::masterTestHolder::getNumLibs ( ) const `[inline]`

Number of libraries in the set.

Returns

test::masterTestHolder::libraryList.size()

int test::masterTestHolder::getNumRun ( ) const `[inline]`

Number of libraries attempted to run.

Returns

test::masterTestHolder::libsRun (p. 450)

int test::masterTestHolder::getNumSuccess ( ) const [inline]

Number of libraries successfully completed.

Returns

test::masterTestHolder::libsCompleted (p. 450)

void test::masterTestHolder::pushLibrary (os::smart_ptr< **libraryTests** > lib) [inline]

Add library to the set.

Adds a **test::libraryTests** (p. 440) to the set of library tests to be tested.

Parameters

in	*lib*	Library test to be added to set

Returns

void

void test::masterTestHolder::removeLibrary (os::smart_ptr< **libraryTests** > lib) [inline]

Remove library from the set.

Removes a **test::libraryTests** (p. 440) from the set of library tests to be tested.

Parameters

in	*lib*	Library test to be removed from the set

Returns

void

bool test::masterTestHolder::runTests ( ) throw os::smart_ptr< std::exception >)

Runs all of the library tests.

Runs all library tests bound to this class. Each library should manage its own errors, but it is possible that this function will throw an error of type os::smart_ptr<std::exception>.

Returns

True if all the tests were successful, else, false

static os::smart_ptr<**masterTestHolder**> test::masterTestHolder::singleton ( ) [static]

Singleton access.

This function constructs the single reference to the **test::masterTestHolder** (p. 447) class if needed. Then, it returns a pointer to this single reference.

Returns

Singleton reference to **test::masterTestHolder** (p. 447)

18.3.4 Member Data Documentation

os::smartSet<**libraryTests**> test::masterTestHolder::libraryList `[private]`

Set of library tests.

int test::masterTestHolder::libsCompleted `[private]`

Number of libraries successfully completed.

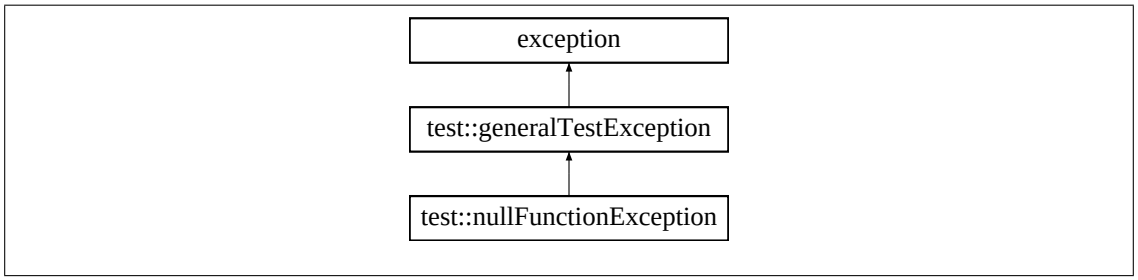
int test::masterTestHolder::libsRun `[private]`

Number of libraries attempted to run.

18.4 test::nullFunctionException Class Reference

NULL function exception class.

Inheritance diagram for test::nullFunctionException:



Public Member Functions

- **nullFunctionException** (std::string loc)

  Construct exception with location.
- virtual ~**nullFunctionException** () throw ()

  Virtual destructor.

18.4.1 Detailed Description

NULL function exception class.

This class defines the common exception case where a NULL function pointer is received.

18.4.2 Constructor & Destructor Documentation

test::nullFunctionException::nullFunctionException (std::string loc) `[inline]`

Construct exception with location.

Constructs a **test::generalTestException** (p. 438) with the provided location and the static string for a NULL function exception.

Parameters

`in`	*loc*	Location string

virtual test::nullFunctionException::~nullFunctionException ( ) throw) `[inline]`, `[virtual]`

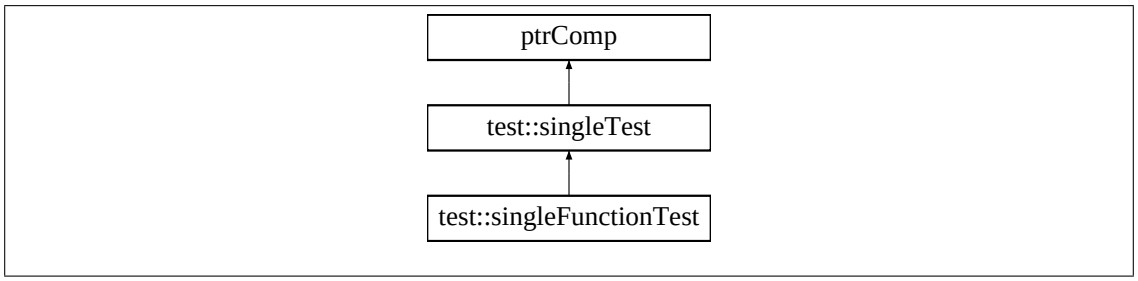
Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.5 test::singleFunctionTest Class Reference

Single unit test from function.

Inheritance diagram for test::singleFunctionTest:



Public Member Functions

- **singleFunctionTest** (std::string tn, **testFunction** f)

  Single unit test constructor.
- virtual ~**singleFunctionTest** ()

  Virtual destructor.
- void **test** () throw (os::smart_ptr<std::exception>)

  Call unit test function.

Private Attributes

- **testFunction func**

  Reference to unit test function.

18.5.1 Detailed Description

Single unit test from function.

This class allows a **test::singleTest** (p. 452) to be defined by a single test function.

18.5.2 Constructor & Destructor Documentation

test::singleFunctionTest::singleFunctionTest (std::string tn, **testFunction** f)

Single unit test constructor.

Parameters

in	*tn*	Name of unit test
in	*f*	Function which defines test

virtual test::singleFunctionTest::~singleFunctionTest ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.5.3 Member Function Documentation

void test::singleFunctionTest::test ( ) throw os::smart_ptr< std::exception >) [virtual]

Call unit test function.

Calls the function bound to this class in the constructor pointed to by **test::singleFunctionTest↩ ::func** (p. 452). If the function pointed to by the function pointer throws an exception, this function will throw the same exception.

Returns

void

Reimplemented from **test::singleTest** (p. 455).

18.5.4 Member Data Documentation

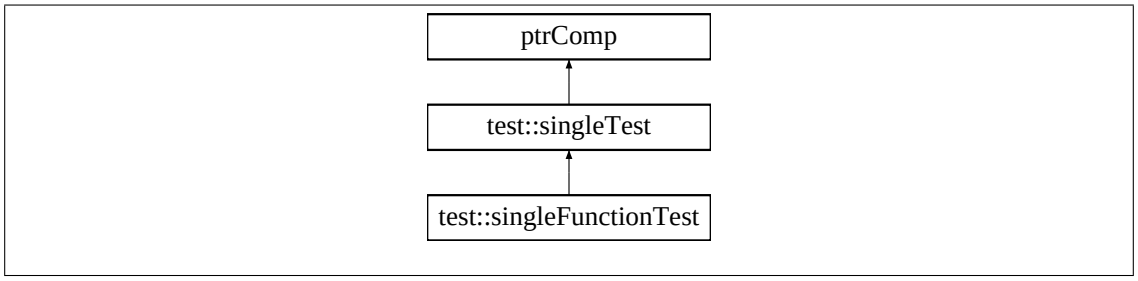
testFunction test::singleFunctionTest::func [private]

Reference to unit test function.

18.6 test::singleTest Class Reference

Single unit test class.

Inheritance diagram for test::singleTest:



Public Member Functions

- **singleTest** (std::string tn)

  Single unit test constructor.
- virtual ~**singleTest** ()

  Virtual destructor.
- virtual void **setupTest** () throw (os::smart_ptr<std::exception>)

  Preforms any test set-up.
- virtual void **test** () throw (os::smart_ptr<std::exception>)

  Preforms core unit-test.
- virtual void **teardownTest** () throw (os::smart_ptr<std::exception>)

  Preforms any test tear-down.
- void **logBegin** ()

  Prints out the name of the test.
- bool **logEnd** (os::smart_ptr< std::exception > except=NULL)

  Logs errors for test.

Private Attributes

- std::string **testName**

  Name of unit test.

18.6.1 Detailed Description

Single unit test class.

This class acts as the base class for all unit tests. It inherits from the os::ptrComp class to allow it to be inserted into abstract data-structures.

18.6.2 Constructor & Destructor Documentation

test::singleTest::singleTest (std::string tn)

Single unit test constructor.

Parameters

in	*tn*	Name of unit test

virtual test::singleTest::~singleTest ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.6.3 Member Function Documentation

void test::singleTest::logBegin ( )

Prints out the name of the test.

Returns

void

bool test::singleTest::logEnd (os::smart_ptr< std::exception > except = NULL)

Logs errors for test.

If the passed exception is NULL, no logging is preformed. Otherwise, the "what()" function of the exception is printed. This function return true if NULL is passed as the exception.

Parameters

in	*except*	Exception to be printed, NULL by default

Returns

True if except is NULL

virtual void test::singleTest::setupTest ( ) throw os::smart_ptr< std::exception >) [inline], [virtual]

Preforms any test set-up.

This function is designed to preform any set-up a test requires. This is especially useful if a class of tests require the same set-up routine. This function assumes that the **test::testSuite** (p. 455) will catch exceptions in this function if they are thrown.

Returns

void

virtual void test::singleTest::teardownTest ( ) throw os::smart_ptr< std::exception >) [inline], [virtual]

Preforms any test tear-down.

This function is designed to preform any tear-down a test requires. This is especially useful if a class of tests require the same tear-down routine. This function assumes that the **test::testSuite** (p. 455) will catch exceptions in this function if they are thrown.

Returns

void

virtual void test::singleTest::test ( ) throw os::smart_ptr< std::exception >) [virtual]

Preforms core unit-test.

This function is designed to preform the actual unit test. This function assumes that the **test::testSuite** (p. 455) will catch exceptions in this function if they are thrown.

Returns

void

Reimplemented in **test::singleFunctionTest** (p. 452).

18.6.4 Member Data Documentation

std::string test::singleTest::testName [private]

Name of unit test.

18.7 test::testSuite Class Reference

Public Member Functions

- **testSuite** (std::string sn)
  Test suite constructor.
- virtual ~**testSuite** ()
  Virtual destructor.
- void **runTests** () throw (os::smart_ptr<std::exception>)
  Runs all of the tests.
- virtual void **onSetup** ()
  Runs on shutdown.
- virtual void **onTeardown** ()
  Runs on teardown of the suite.
- void **logBegin** ()
  Logs the beginning of a suite test.
- bool **logEnd** (os::smart_ptr< std::exception > except=NULL)
  Logs the end of a suite test.
- int **getNumTests** () const
  Number of tests in the set.
- int **getNumSuccess** () const
  Number of tests successfully completed.
- int **getNumRun** () const
  Number of tests attempted to run.
- void **pushTest** (os::smart_ptr< **singleTest** > tst)

Add test to the set.

- void **removeTest** (os::smart_ptr< **singleTest** > tst)

  Remove test to the set.
- virtual void **pushTest** (std::string str, **testFunction** tst)

  Add test to the set.
- bool **operator==** (const **testSuite** &lt) const

  Equality comparison.
- bool **operator!=** (const **testSuite** &lt) const

  Not-equals comparison.
- bool **operator**> (const **testSuite** &lt) const

  Greater-than comparison.
- bool **operator**< (const **testSuite** &lt) const

  Less-than comparison.
- bool **operator**>**=** (const **testSuite** &lt) const

  Greater-than or equal to comparison.
- bool **operator**<**=** (const **testSuite** &lt) const

  Less-than or equal to comparison.

Private Attributes

- std::string **suiteName**

  Name of test suite.
- os::smartSet< **singleTest** > **testList**

  Set of tests.
- int **testsCompleted**

  Number of tests successfully completed.
- int **testsRun**

  Number of tests attempted to run.

18.7.1 Constructor & Destructor Documentation

test::testSuite::testSuite (std::string sn)

Test suite constructor.

This constructor initializes the number of tests completed and number of tests run to 0, along with sets the name of suite being tested.

Parameters

`in`	*sn*	Name of suite to be tested

virtual test::testSuite::~testSuite ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

18.7.2 Member Function Documentation

int test::testSuite::getNumRun ( ) const `[inline]`

Number of tests attempted to run.

Returns

test::testSuite::testsRun (p. 461)

int test::testSuite::getNumSuccess ( ) const `[inline]`

Number of tests successfully completed.

Returns

test::testSuite::testsCompleted (p. 461)

int test::testSuite::getNumTests ( ) const `[inline]`

Number of tests in the set.

Returns

test::testSuite::testList.size()

void test::testSuite::logBegin ( )

Logs the beginning of a suite test.

Outputs the name of the suite to be tested along with a line break made of '-' characters.

Returns

void

bool test::testSuite::logEnd (os::smart_ptr< std::exception > except = `NULL`)

Logs the end of a suite test.

Outputs the number of tests run and how many of these tests were both successful and how many of these tests failed.

Returns

True if all tests successful, else false

virtual void test::testSuite::onSetup ( ) `[inline]`, `[virtual]`

Runs on shutdown.

Each suite calls this funciton as it starts up, allowing suites to define actions performed to setup the suite.

Returns

void

virtual void test::testSuite::onTeardown ( ) [inline], [virtual]

Runs on teardown of the suite.

Guranteed to run even if the suite itself fails. A custom tear-down for the suite can re-impliment this class.

Returns

void

bool test::testSuite::operator!= (const **testSuite** & lt) const [inline]

Not-equals comparison.

Compares two **test::testSuite** (p. 455) based on the library name. If the two names are not-equal, the suites are not-equal.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName!=lt.suiteName

bool test::testSuite::operator< (const **testSuite** & lt) const [inline]

Less-than comparison.

Compares two **test::testSuite** (p. 455) based on the library name. If the name of this object is less than the name of the reference object, return true.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName<lt.suiteName

bool test::testSuite::operator<= (const **testSuite** & lt) const [inline]

Less-than or equal to comparison.

Compares two **test::testSuite** (p. 455) based on the library name. If the name of this object is less than or equal to the name of the reference object, return true.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName<=lt.suiteName

bool test::testSuite::operator== (const **testSuite** & lt) const [inline]

Equality comparison.

Compares two **test::testSuite** (p. 455) based on the suite name. If the two names are equal, the suites are equal.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName==lt.suiteName

bool test::testSuite::operator> (const **testSuite** & lt) const [inline]

Greater-than comparison.

Compares two **test::testSuite** (p. 455) based on the library name. If the name of this object is greater than the name of the reference object, return true.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName>lt.suiteName

bool test::testSuite::operator>= (const **testSuite** & lt) const [inline]

Greater-than or equal to comparison.

Compares two **test::testSuite** (p. 455) based on the library name. If the name of this object is greater than or equal to the name of the reference object, return true.

Parameters

in	*lt*	Reference to **test::testSuite** (p. 455) to be compared against

Returns

this->suiteName>=lt.suiteName

void test::testSuite::pushTest (os::smart_ptr< **singleTest** > tst) [inline]

Add test to the set.

Adds a **test::singleTest** (p. 452) to the set of tests to be tested.

Parameters

in	*tst*	Test to be added to set

Returns

void

virtual void test::testSuite::pushTest (std::string str, **testFunction** tst) [inline], [virtual]

Add test to the set.

Adds a **test::testFunction** (p. 436) to the set of tests to be tested. Constructs a **test::singleTest** (p. 452) from a function and a test name

Parameters

in	*str*	Test name
in	*tst*	Function which defines test

Returns

void

void test::testSuite::removeTest (os::smart_ptr< **singleTest** > tst) [inline]

Remove test to the set.

Removes a **test::singleTest** (p. 452) from the set of tests to be tested.

Parameters

in	*tst*	Test to be removed from the set

Returns

void

void test::testSuite::runTests ( ) throw os::smart_ptr< std::exception >)

Runs all of the tests.

Runs all tests bound to this class. This function catches exceptions thrown by **test::singleTest** (p. 452) and logs the results.

Returns

void

18.7.3 Member Data Documentation

std::string test::testSuite::suiteName [private]

Name of test suite.

os::smartSet<**singleTest**> test::testSuite::testList [private]

Set of tests.

int test::testSuite::testsCompleted [private]

Number of tests successfully completed.

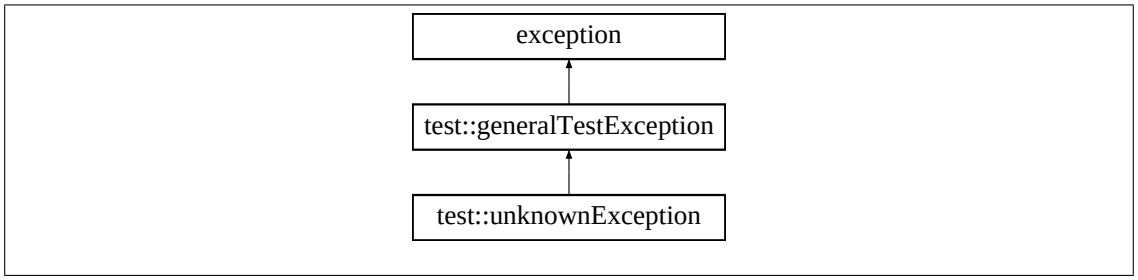
int test::testSuite::testsRun [private]

Number of tests attempted to run.

18.8 test::unknownException Class Reference

Unknown exception class.

Inheritance diagram for test::unknownException:



Public Member Functions

- **unknownException** (std::string loc)

  Construct exception with location.
- virtual ~**unknownException** () throw ()

  Virtual destructor.

18.8.1 Detailed Description

Unknown exception class.

This class defines the common exception case where the precise nature of the exception is unknown.

18.8.2 Constructor & Destructor Documentation

test::unknownException::unknownException (std::string loc) [inline]

Construct exception with location.

Constructs a **test::generalTestException** (p. 438) with the provided location and the static string for an unknown exception.

Parameters

in	*loc*	Location string

virtual test::unknownException::~unknownException ( ) throw) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

Part IV

osMechanics Library

Chapter 19

Introduction

The osMechanics library contains classes which are general tools for navigating file systems, thread management and logging. Some classes, particularly those dealing with threading, sockets and file access, differ from operating system to operating system. CMake should handle all operating system variances.

19.1 Namespace

osMechanics extends the os namespace. The os namespace is designed for tools, algorithms and data-structures used in programs of all types. Note that the Datastructures library also uses the os namespace.

Chapter 20

File Index

20.1 File List

Here is a list of all files with brief descriptions:

Chapter 21

File Documentation

21.1 logger.cpp File Reference

logger implementation file

Functions

- static void **loggerSavingThread** (void *ptr, smart_ptr< **threadHolder** > th)

Variables

- smart_ptr< **Log** > **_single_log**
- static bool **singleton_bool** = false

21.1.1 Detailed Description

logger implementation file
Jonathan Bedard

Date

4/23/2015

Bug No known bugs.

The implementation of our logging systems are in this file. The logger records various timing, operation, and debug information and places it in various files so that we can better analyze our own system's performance.

21.1.2 Function Documentation

static void loggerSavingThread (void * ptr, smart_ptr< **threadHolder** > th) [static]

21.1.3 Variable Documentation

smart_ptr<**Log**> _single_log

bool singleton_bool = false [static]

21.2 logger.h File Reference

logger header file

Classes

- class **os::logStatusHolder**
- class **os::logStatusListener**
- struct **os::logLine**
- class **os::LogStreamListener**
- class **os::LineLogger**
- class **os::LogSaver**
- class **os::LineSaver**
- class **os::LineSaverListener**
- class **os::Log**
- class **os::LogDirectedStream**

Namespaces

- **os**

Variables

- **logStatusHolder os::logStatus**
- **Log** & **os::logger** =∗Log::singleton()

21.2.1 Detailed Description

logger header file
Jonathan Bedard

Date

4/23/2016

Bug No known bugs.

All of the headers in the Datastructures library are held in this file. When using the Datastructures library, it is expected that this header is included instead of the individual required headers.

21.3 multiLock.cpp File Reference

multiLock implementation file

21.3.1 Detailed Description

multiLock implementation file
Jonathan Bedard

Date

9/29/2015

Bug No known bugs.

This is the implementation of our multiLock. It is platform agnostic.

21.4 multiLock.h File Reference

multiLock header file

Classes

- class **os::multiLock**
  os::multilock class definition Defines the os::multilock class. This class has 4 variables and 8 methods

Namespaces

- **os**

21.4.1 Detailed Description

multiLock header file
Jonathan Bedard

Date

1/30/2016

Bug No known bugs.

This is the multilock header we are using. It has reading and writing locks, allowing multiple users to read, but only one to write at any given time.

21.5 osFunctions.cpp File Reference

21.6 osFunctions.cpp File Reference

Functions

- static void **receiveThreadServerIPV4** (void ∗ptr, smart_ptr< **threadHolder** > th)
- static void **receiveThreadServerIPV6** (void ∗ptr, smart_ptr< **threadHolder** > th)

Variables

- static os::smart_ptr< **threadDistributor** > **ipthread** = NULL
- static std::string **local_path** = ""

21.6.1 Function Documentation

static void receiveThreadServerIPV4 (void * ptr, smart_ptr< **threadHolder** > th) [static]

static void receiveThreadServerIPV6 (void * ptr, smart_ptr< **threadHolder** > th) [static]

21.6.2 Variable Documentation

os::smart_ptr<**threadDistributor**> ipthread = NULL [static]

std::string local_path = "" [static]

21.7 osFunctions.cpp File Reference

21.8 osFunctions.h File Reference

Namespaces

- **os**

Functions

- uint16_t **os::to_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint16_t **os::from_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint32_t **os::to_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint32_t **os::from_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint64_t **os::to_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint64_t **os::from_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint64_t **os::getTimestamp** ()

  Gets a timestamp Generates a time stamp from the time function.
- bool **os::testCreateFolder** (std::string n)

Test if a folder exists Checks if a given folder exists. If it does not exist, this function will create said folder.

- std::string **os::convertTimestamp** (uint64_t stamp)

  Type conversion on timestamp Converts the timestamp from an integer into a string.

21.9 osFunctions.h File Reference

Classes

- class **os::IPAddress**

  os::IPAddress *(p. 497) class definition This is an IP Address class It has 2 variables and 10 methods*
- class **os::myIPAddress**

  Holds a node's own IP address Every node needs it's own IP address. This class holds that value, as well as provide several functions for determining priorities.
- class **os::UDPPacket**
- class **os::UDPClient**
- struct **os::UDPAVLNode**
- class **os::UDPServer**

Namespaces

- **os**

Functions

- void **os::sleep** (int32_t x)

  Sleep the thread for a certain amount of time This is a simple sleep function, it takes in a length of time to sleep and return nothing.
- void **os::startInternet** (bool multiThread=true)

  Activates Internet Spawns an IP thread distributor, if one does not currently exist.
- void **os::closeInternet** ()

  Deactivates Internet deletes the IP thread distributor and sets the thread pointer to null.
- smart_ptr< **threadDistributor** > **os::internetThreads** ()

  Return IP thread distributor Gives the ipthread distributor to the caller.
- int32_t **os::cp_clock_gettime** (int32_t X, struct timeval *tv)

  Gets time Returns the current time to the caller. This is designed to work across a range of platforms and format the time to a high precision.
- void **os::strcpy_s** (char *output, int32_t inlen, const char *input)

  String copier Safely calls string copy.
- bool **os::is_directory** (std::string file)

  Determines if a file is a directory Checks if a given file is a directory.
- bool **os::check_exists** (std::string name)

  Checks if a given file exists Takes a file and checks if it exists. A directory is considered existing.
- smart_ptr< std::string > **os::list_files** (std::string directory, uint32_t &len)

Return contents of directory Creates an array of strings of all the names inside a given directory. This is not recursive.

- std::string **os::extract_name** (std::string full_path)

  Extracts a given file Extracts a file or directory.
- void **os::delete_file** (std::string path)

  Deletes a file Deletes the file or directory at the given path. This is a recursive delete.
- void **os::setLocalPath** (int argc, char **argv)

  Sets local path Sets the local path given the received arguments.
- std::string **os::getLocalPath** ()

  Returns local path.
- static int32_t **os::fopen_s** (FILE **fp, const char *file_name, const char *typ)

  fopen_s for windows This is a file open function for windows so that we can more efficiently write multi platform code.

Variables

- const uint32_t **CLOCK_REALTIME** =0
- const uint32_t **CLOCK_MONOTONIC** =1
- const uint32_t **os::BUFLEN** =512
- const std::string **os::DEFAULT_IP** ="127.0.0.1"
- const uint32_t **os::MY_MESSAGE_NOTIFICATION** =1048

21.9.1 Variable Documentation

const uint32_t CLOCK_MONOTONIC =1

const uint32_t CLOCK_REALTIME =0

21.10 osFunctions.h File Reference

21.11 osMechanics.h File Reference

osMechanics header file

21.11.1 Detailed Description

osMechanics header file
Jonathan Bedard

Date

2/24/2015

Bug No known bugs.

This file includes all of our headers, so that other libraries can easily include the osMechanics library with one include.

21.12 osMechanicsTest.cpp File Reference

Test implimentaiton for osMechanics.

21.12.1 Detailed Description

Test implimentaiton for osMechanics.

Author

Adrian Bedard

Date

4/12/2016

Bug No known bugs.

Binds all osMechanics test suites. These suites test the basic funcitonality of the osMechanics library. Projects which utilize osMechanics are suggested to bind the osMechanics library tests to their own test suite.

21.13 osMechanicsTest.h File Reference

osMechanics tests

21.13.1 Detailed Description

osMechanics tests
Jonathan Bedard

Date

4/11/2016

Bug No known bugs.

This is the test suite for the osMechanics library.

21.14 osThreads.cpp File Reference

threads implementation file

Functions

- void **temp_thread_call** (void *ptr_array, bool typ, std::string thread_info)
- void **wait_for_threads** ()

Variables

- static **spinLock globalThreadLock**
- static **threadTracker** * **static_ref** = NULL

21.14.1 Detailed Description

threads implementation file
Jonathan Bedard

Date

4/18/2016

Bug No known bugs.

This is the implementation of our multi threading system.

21.14.2 Function Documentation

void temp_thread_call (void * ptr_array, bool typ, std::string thread_info)

void wait_for_threads ( )

21.14.3 Variable Documentation

spinLock globalThreadLock [static]

threadTracker* static_ref = NULL [static]

21.15 osThreads.h File Reference

osThreads header file

Classes

- class **os::threadHolder**
- class **os::threadTracker**

  Monitors a range of threads This class holds a range of threadHolders. This includes both active and expired threads, ensuring the ability to operate on many threads in mass.

Namespaces

- **os**

Functions

- smart_ptr< std::thread > **os::spawnThread** (void(*func)(void *), void *ptr, std::string thread_info="")
- smart_ptr< std::thread > **os::spawnThread** (void(*func)(void *, smart_ptr< **threadHolder** >), void *ptr, std::string thread_info="")

21.15.1 Detailed Description

osThreads header file
Jonathan Bedard

Date

4/13/2016

Bug No known bugs.

This is the osThreads header we are using. This header allows us to use multithreading with our own types, pointers, and management

21.16 safeQueue.h File Reference

safe queue header file

Classes

- class **os::safeQueue**< **dataType** >

  *This is the **safeQueue** (p. 519) class The **safeQueue** (p. 519) class is thread safe. It is a template class.*

Namespaces

- **os**

21.16.1 Detailed Description

safe queue header file
Jonathan Bedard

Date

11/9/2015

Bug No known bugs.

This is a thread safe queue, so we can multi thread safely.

21.17 savableClass.cpp File Reference

Implementation of the generalized savable class.

21.17.1 Detailed Description

Implementation of the generalized savable class.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

Provides an implementation of the savable class, used to tie together multiple classes which need to be saved as a group.

21.18 savableClass.h File Reference

Defines a set of classes facilitating saving.

Classes

- class **os::savable**
  Basic saving class.
- class **os::savingGroup**
  Group of saving classes.

Namespaces

- **os**

21.18.1 Detailed Description

Defines a set of classes facilitating saving.

Author

Jonathan Bedard

Date

4/12/2016

Bug None

Provides a definition of user which has a user-name, password and associated bank of public keys.

21.19 Serial.h File Reference

21.20 Serial.h File Reference

Classes

- class **os::Serial**

  *This is the **Serial** (p. 528) class. **Serial** (p. 528) objects allow us to abstract out most of the platform irregularities across multiple systems.*

Namespaces

- **os**

Variables

- const uint32_t **os::ARDUINO_WAIT_TIME** =2000

21.21 Serial.h File Reference

21.22 serialThread.cpp File Reference

serialThread implementation file

Functions

- static void **serialSearch** (void *ptr, smart_ptr< **threadHolder** > th)

21.22.1 Detailed Description

serialThread implementation file
Jonathan Bedard

Date

11/1/2015

Bug No known bugs.

These implementations allow us to create threads for monitoring serial communication

21.22.2 Function Documentation

static void serialSearch (void * ptr, smart_ptr< **threadHolder** > th) [static]

21.23 serialThread.h File Reference

serial thread header file

Classes

- class **os::serialThread**

  Serial *(*p. *528) communication thread The is a serial class that runs as a thread. Thanks to this fact, we can run multiple serial communication threads as well as run a primary set of threads at once.*

Namespaces

- **os**

21.23.1 Detailed Description

serial thread header file
Jonathan Bedard

Date

11/9/2015

Bug No known bugs.

This is a serial thread class. This class allows us to monitor multiple ports effectively simultaneously.

21.24 socketFrame.cpp File Reference

socketFrame implementation file

Functions

- void **close_all_sockets** ()

Variables

- static smart_ptr< **socketTracker** > **st_instance** = NULL

21.24.1 Detailed Description

socketFrame implementation file
Jonathan Bedard

Date

2/12/2016

Bug No known bugs.

This is the implementation of our socket user, UDP socket, and socket tracker. Socket communication is important for us, and this allows us to safely have reliable sockets.

21.24.2 Function Documentation

void close_all_sockets ( )

21.24.3 Variable Documentation

smart_ptr<**socketTracker**> st_instance = NULL `[static]`

21.25 socketFrame.h File Reference

socket frame header file

Classes

- class **os::socketUser**

  Socket user class This class allows us to manage sockets.
- class **os::UDPSocket**

  UDPSocket *(*p. *559) class A class for UDPSockets, which in turn allows us to multi thread the packet send/receive functionality.*
- class **os::socketTracker**

  socketTracker *(*p. *535) class Tracks all currently active sockets.*

Namespaces

- **os**

21.25.1 Detailed Description

socket frame header file
Jonathan Bedard

Date

4/12/2016

Bug No known bugs.

Generalized socket class.

21.26 spinLock.cpp File Reference

21.27 spinLock.cpp File Reference

21.28 spinLock.cpp File Reference

21.29 spinLock.h File Reference

21.30 spinLock.h File Reference

Classes

- class **os::spinLock**

Namespaces

- **os**

21.31 spinLock.h File Reference

21.32 threadDistribution.cpp File Reference

thread distribution implementation file

Functions

- static void **executor_thread_starter** (void *ptr, smart_ptr< **threadHolder** > th)

21.32.1 Detailed Description

thread distribution implementation file
Jonathan Bedard

Date

4/18/2015

Bug No known bugs.

These methods determine which thread will operate next.

21.32.2 Function Documentation

static void executor_thread_starter (void * ptr, smart_ptr< **threadHolder** > th) [static]

21.33 threadDistribution.h File Reference

thread distribution header file

Classes

- class **os::threadActor**

  threadActor *(p. 539) class This class holds information for determining which thread goes at a give time.*
- class **os::threadDistributor**

  Distributes threads This class allows us to determine which thread should execute at any given time.

- class **os::executorThread**

  executorThread *(*p. *495) class This class holds a thread which has multiple steps.*
- class **os::singleAction**

  single action class This class is for a thread with only one action.

Namespaces

- **os**

Functions

- float **os::getSysTime** ()

  gets time Gets the current system time.

21.33.1 Detailed Description

thread distribution header file
Jonathan Bedard

Date

4/18/2015

Bug No known bugs.

This the thread distribution system.

21.34 USBAccess.cpp File Reference

USBAccess implementation file.

21.34.1 Detailed Description

USBAccess implementation file.
Jonathan Bedard

Date

11/3/2015

Bug No known bugs.

These are simple USB methods. They are unused in our larger project.

21.35 USBAccess.h File Reference

USBAccess header file.

Classes

- class **os::USBNode**

  This class stores the location of a USB device.
- class **os::USBFile**

Namespaces

- **os**

21.35.1 Detailed Description

USBAccess header file.
Jonathan Bedard

Date

6/21/2015

Bug No known bugs.

This is a pair of simple classes for working with USB devices.

21.36 XMLParser.cpp File Reference

XML parser implementation file.

21.36.1 Detailed Description

XML parser implementation file.
Jonathan Bedard

Date

2/7/2015

Bug No known bugs.

Our XML parse is implemented in this file. We have several functions that allow us to easily convert XML data from file to program and vice versa.

21.37 XMLParser.h File Reference

XML Parser header file.

Classes

- class **os::XML_Node**

  XML Node class The core node of our XML parsing.

Namespaces

- **os**
- **os::xml**

Typedefs

- typedef smart_ptr< XML_Node > **os::smartXMLNode**
- typedef smart_ptr< unsortedList< XML_Node > > **os::smartXMLNodeList**

Functions

- bool **os::XML_Output** (std::string path, **smartXMLNode** head)

  outputs tree Outputs an XML tree into a file.
- **smartXMLNode os::XML_Input** (std::string path)

  imports tree Imports an XML tree from a file.
- void **os::xml::insertTabs** (std::ofstream &f, int32_t x)

  adds tabs Adds tabs.
- void **os::xml::writeNode** (std::ofstream &f, smartXMLNode node, int32_t depth)

  writes nodes Writes all the nodes to a file. This function runs recursively.
- std::vector< std::string > **os::xml::readTillTag** (std::ifstream &f)

  reads until next tag Reads a file until the next tag is found.
- std::string **os::xml::readThroughTag** (std::ifstream &f)

  reads through the next tag Reads a file until a tag is found, including that tag.
- **smartXMLNode os::xml::parseNode** (std::ifstream &f)

  parses a node Pulls a node from a file and returns it.
- bool **os::xml::compareTrees** (**smartXMLNode** n1, **smartXMLNode** n2)

  compares trees Determines if two nodes are equivalent.

21.37.1 Detailed Description

XML Parser header file.
Jonathan Bedard

Date

2/7/2015

Bug No known bugs.

This is our XML Parser, so we can standardize use across systems.

21.38 XMLTest.cpp File Reference

XML tests.

21.38.1 Detailed Description

XML tests.
Jonathan Bedard

Date

2/29/2016

Bug No known bugs.

These are the tests for our XML classes.

21.39 XMLTest.h File Reference

SML test header file.

21.39.1 Detailed Description

SML test header file.
Jonathan Bedard

Date

4/12/2016

Bug No known bugs.

This is the test suite for the XML tests.

Chapter 22

Class Index

22.1 Class List

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

Chapter 23

Namespace Documentation

23.1 os Namespace Reference

Namespaces

- **xml**

Classes

- class **executorThread**

  executorThread *(p. 495) class This class holds a thread which has multiple steps.*
- class **IPAddress**

  os::IPAddress *(p. 497) class definition This is an IP Address class It has 2 variables and 10 methods*
- class **LineLogger**
- class **LineSaver**
- class **LineSaverListener**
- class **Log**
- class **LogDirectedStream**
- struct **logLine**
- class **LogSaver**
- class **logStatusHolder**
- class **logStatusListener**
- class **LogStreamListener**
- class **multiLock**

  os::multilock class definition Defines the os::multilock class. This class has 4 variables and 8 methods
- class **myIPAddress**

  Holds a node's own IP address Every node needs it's own IP address. This class holds that value, as well as provide several functions for determining priorities.
- class **safeQueue**

  This is the ***safeQueue*** *(p. 519) class The* ***safeQueue*** *(p. 519) class is thread safe. It is a template class.*
- class **savable**

  Basic saving class.

- class **savingGroup**

  Group of saving classes.
- class **Serial**

  *This is the **Serial** (p. 528) class. **Serial** (p. 528) objects allow us to abstract out most of the platform irregularities across multiple systems.*
- class **serialThread**

  ***Serial** (p. 528) communication thread The is a serial class that runs as a thread. Thanks to this fact, we can run multiple serial communication threads as well as run a primary set of threads at once.*
- class **singleAction**

  single action class This class is for a thread with only one action.
- class **socketTracker**

  ***socketTracker** (p. 535) class Tracks all currently active sockets.*
- class **socketUser**

  Socket user class This class allows us to manage sockets.
- class **spinLock**
- class **threadActor**

  ***threadActor** (p. 539) class This class holds information for determining which thread goes at a give time.*
- class **threadDistributor**

  Distributes threads This class allows us to determine which thread should execute at any given time.
- class **threadHolder**
- class **threadTracker**

  Monitors a range of threads This class holds a range of threadHolders. This includes both active and expired threads, ensuring the ability to operate on many threads in mass.
- struct **UDPAVLNode**
- class **UDPClient**
- class **UDPPacket**
- class **UDPServer**
- class **UDPSocket**

  ***UDPSocket** (p. 559) class A class for UDPSockets, which in turn allows us to multi thread the packet send/receive functionality.*
- class **USBFile**
- class **USBNode**

  This class stores the location of a USB device.
- class **XML_Node**

  XML Node class The core node of our XML parsing.

Typedefs

- typedef smart_ptr< **XML_Node** > **smartXMLNode**
- typedef smart_ptr< unsortedList< **XML_Node** > > **smartXMLNodeList**

Functions

- uint16_t **to_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint16_t **from_comp_mode** (uint16_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint32_t **to_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint32_t **from_comp_mode** (uint32_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint64_t **to_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.
- uint64_t **from_comp_mode** (uint64_t i)

  Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.
- uint64_t **getTimestamp** ()

  Gets a timestamp Generates a time stamp from the time function.
- bool **testCreateFolder** (std::string n)

  Test if a folder exists Checks if a given folder exists. If it does not exist, this function will create said folder.
- std::string **convertTimestamp** (uint64_t stamp)

  Type conversion on timestamp Converts the timestamp from an integer into a string.
- smart_ptr< std::thread > **spawnThread** (void(∗func)(void ∗), void ∗ptr, std::string thread_↩ info="")
- smart_ptr< std::thread > **spawnThread** (void(∗func)(void ∗, smart_ptr< **threadHolder** >), void ∗ptr, std::string thread_info="")
- float **getSysTime** ()

  gets time Gets the current system time.
- bool **XML_Output** (std::string path, **smartXMLNode** head)

  outputs tree Outputs an XML tree into a file.
- **smartXMLNode XML_Input** (std::string path)

  imports tree Imports an XML tree from a file.
- void **sleep** (int32_t x)

  Sleep the thread for a certain amount of time This is a simple sleep function, it takes in a length of time to sleep and return nothing.
- void **startInternet** (bool multiThread=true)

  Activates Internet Spawns an IP thread distributor, if one does not currently exist.
- void **closeInternet** ()

  Deactivates Internet deletes the IP thread distributor and sets the thread pointer to null.
- smart_ptr< **threadDistributor** > **internetThreads** ()

Return IP thread distributor Gives the ipthread distributor to the caller.

- int32_t **cp_clock_gettime** (int32_t X, struct timeval *tv)

  Gets time Returns the current time to the caller. This is designed to work across a range of platforms and format the time to a high precision.
- void **strcpy_s** (char *output, int32_t inlen, const char *input)

  String copier Safely calls string copy.
- bool **is_directory** (std::string file)

  Determines if a file is a directory Checks if a given file is a directory.
- bool **check_exists** (std::string name)

  Checks if a given file exists Takes a file and checks if it exists. A directory is considered existing.
- smart_ptr< std::string > **list_files** (std::string directory, uint32_t &len)

  Return contents of directory Creates an array of strings of all the names inside a given directory. This is not recursive.
- std::string **extract_name** (std::string full_path)

  Extracts a given file Extracts a file or directory.
- void **delete_file** (std::string path)

  Deletes a file Deletes the file or directory at the given path. This is a recursive delete.
- void **setLocalPath** (int argc, char **argv)

  Sets local path Sets the local path given the received arguments.
- std::string **getLocalPath** ()

  Returns local path.
- static int32_t **fopen_s** (FILE **fp, const char *file_name, const char *typ)

  fopen_s for windows This is a file open function for windows so that we can more efficiently write multi platform code.

Variables

- **logStatusHolder logStatus**
- **Log** & **logger** =***Log::singleton**()
- const uint32_t **BUFLEN** =512
- const std::string **DEFAULT_IP** ="127.0.0.1"
- const uint32_t **MY_MESSAGE_NOTIFICATION** =1048
- const uint32_t **ARDUINO_WAIT_TIME** =2000

23.1.1 Typedef Documentation

typedef smart_ptr<**XML_Node**> **os::smartXMLNode**

typedef smart_ptr<unsortedList<**XML_Node**> > **os::smartXMLNodeList**

23.1.2 Function Documentation

bool os::check_exists (std::string name)

Checks if a given file exists Takes a file and checks if it exists. A directory is considered existing.

Returns

bool

void os::closeInternet ( )

Deactivates Internet deletes the IP thread distributor and sets the thread pointer to null.

std::string os::convertTimestamp (uint64_t stamp)

Type conversion on timestamp Converts the timestamp from an integer into a string.

Returns

std::string

int32_t os::cp_clock_gettime (int32_t X, struct timeval * tv)

Gets time Returns the current time to the caller. This is designed to work across a range of platforms and format the time to a high precision.

Returns

unit32_t

void os::delete_file (std::string path)

Deletes a file Deletes the file or directory at the given path. This is a recursive delete.

std::string os::extract_name (std::string full_path)

Extracts a given file Extracts a file or directory.

Returns

string

static int32_t os::fopen_s (FILE ** fp, const char * file_name, const char * typ) [static]

fopen_s for windows This is a file open function for windows so that we can more efficiently write multi platform code.

Returns

uint32_t

uint16_t os::from_comp_mode (uint16_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

Returns

uint16_t

uint32_t os::from_comp_mode (uint32_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

Returns

uint32_t

uint64_t os::from_comp_mode (uint64_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps from compatibility mode.

Returns

uint64_t

std::string os::getLocalPath ( )

Returns local path.

Returns

string

float os::getSysTime ( )

gets time Gets the current system time.

Returns

float

uint64_t os::getTimestamp ( )

Gets a timestamp Generates a time stamp from the time function.

Returns

uint64_t

smart_ptr< **threadDistributor** > os::internetThreads ( )

Return IP thread distributor Gives the ipthread distributor to the caller.

Returns

smart_ptr<threadDistributor>

bool os::is_directory (std::string file)

Determines if a file is a directory Checks if a given file is a directory.

Returns

bool

smart_ptr< std::string > os::list_files (std::string directory, uint32_t & len)

Return contents of directory Creates an array of strings of all the names inside a given directory. This is not recursive.

Returns

os::smart_ptr<string>

void os::setLocalPath (int argc, char ** argv)

Sets local path Sets the local path given the received arguments.

void os::sleep (int32_t x)

Sleep the thread for a certain amount of time This is a simple sleep function, it takes in a length of time to sleep and return nothing.

smart_ptr< std::thread > os::spawnThread (void(*)(void *) func, void * ptr, std::string thread_info = "")

smart_ptr< std::thread > os::spawnThread (void(*)(void *, smart_ptr< **threadHolder** >) func, void * ptr, std::string thread_info = "")

void os::startInternet (bool multiThread = `true`)

Activates Internet Spawns an IP thread distributor, if one does not currently exist.

void os::strcpy_s (char * output, int32_t inlen, const char * input)

String copier Safely calls string copy.

bool os::testCreateFolder (std::string n)

Test if a folder exists Checks if a given folder exists. If it does not exist, this function will create said folder.

Returns

bool

uint16_t os::to_comp_mode (uint16_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

Returns

uint16_t

uint32_t os::to_comp_mode (uint32_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

Returns

uint32_t

uint64_t os::to_comp_mode (uint64_t i)

Changes bit order for compatibility Depending on the system at hand, bits may be in several different orders. This function swaps to compatibility mode.

Returns

uint64_t

smartXMLNode os::XML_Input (std::string path)

imports tree Imports an XML tree from a file.

Returns

smart_ptr<XMLNode>

bool os::XML_Output (std::string path, **smartXMLNode** head)

outputs tree Outputs an XML tree into a file.

Returns

bool

23.1.3 Variable Documentation

const uint32_t os::ARDUINO_WAIT_TIME =2000

const uint32_t os::BUFLEN =512

const std::string os::DEFAULT_IP ="127.0.0.1"

Log & os::logger =∗**Log::singleton**()

logStatusHolder os::logStatus

const uint32_t os::MY_MESSAGE_NOTIFICATION =1048

Chapter 24

Class Documentation

24.1 os::executorThread Class Reference

executorThread (p. 495) class This class holds a thread which has multiple steps.

Public Member Functions

- **executorThread** (uint32_t id, smart_ptr< **threadDistributor** > d)
- virtual ~**executorThread** ()
- bool **isRunning** () const

  indicates running Indicates if a thread is currently running.
- uint32_t **getThreadID** () const

  gives thread_id Gives the thread identifier to the caller.
- void **killThread** ()

  kills thread Kills the thread associated with this executor.
- void **primary_thread_loop** (smart_ptr< **threadHolder** > th)

  runs thread Runs the associated thread and manages others.

Private Attributes

- **spinLock killLock**

  lock for killing threads Allows us to kill a thread without changing into it.
- uint32_t **thread_id**

  thread identifier An identifier for the thread.
- volatile bool **active**

  indicates readiness Indicates if a thread is ready to run.
- bool **running**

  indicates running Indicates if a thread is currently running.
- smart_ptr< **threadDistributor** > **distro**

  distributor Indicates the thread distributor for this executor thread.

24.1.1 Detailed Description

executorThread (p. 495) class This class holds a thread which has multiple steps.

24.1.2 Constructor & Destructor Documentation

executorThread::executorThread (uint32_t id, smart_ptr< **threadDistributor** > d)

executorThread::~executorThread ( ) [virtual]

24.1.3 Member Function Documentation

uint32_t os::executorThread::getThreadID ( ) const [inline]

gives thread_id Gives the thread identifier to the caller.

Returns

uint32_t

bool os::executorThread::isRunning ( ) const [inline]

indicates running Indicates if a thread is currently running.

Returns

bool

void executorThread::killThread ( )

kills thread Kills the thread associated with this executor.

void executorThread::primary_thread_loop (smart_ptr< **threadHolder** > th)

runs thread Runs the associated thread and manages others.

24.1.4 Member Data Documentation

volatile bool os::executorThread::active [private]

indicates readiness Indicates if a thread is ready to run.

smart_ptr<**threadDistributor**> os::executorThread::distro [private]

distributor Indicates the thread distributor for this executor thread.

spinLock os::executorThread::killLock [private]

lock for killing threads Allows us to kill a thread without changing into it.

bool os::executorThread::running [private]

indicates running Indicates if a thread is currently running.

uint32_t os::executorThread::thread_id [private]

thread identifier An identifier for the thread.

24.2 os::IPAddress Class Reference

os::IPAddress (p. 497) class definition This is an IP Address class It has 2 variables and 10 methods

Public Member Functions

- **IPAddress** ()
- **IPAddress** (std::string x)
- **IPAddress** (const **IPAddress** &x)
- **IPAddress** (smart_ptr< **IPAddress** > x)
- virtual ~**IPAddress** ()
- bool **isIPv6** () const

  Return if an IPAdress is IPv6 returns the _isIPv6 boolean to the caller.
- char * **printAddress** ()

  *Return IP Address Gives a pointer to the **IPAddress** (p. 497) to the caller.*
- const char * **getConstAddress** () const

  *Return IP Address Gives a pointer to the **IPAddress** (p. 497) to the caller.*
- int32_t **compare** (const os::smart_ptr< **IPAddress** > comp) const

  Compares IP Addresses Compares two IPAddresses and returns the difference.
- int32_t **compare** (const **IPAddress** *comp) const

Protected Attributes

- char **name** [80]

  Actual data of IP address. The actual IP is a critical element, as it allows for the core communication. We allocate 80 bytes so that we have enough memory for both the IPv4 and IPv6 addresses.
- bool **_isIPv6**

  Holds if an address is IPv6 An address has this flag for easy analysis of IPv6 and IPv4 status. Thanks to this, we can use one IP address class for both standards.

24.2.1 Detailed Description

os::IPAddress (p. 497) class definition This is an IP Address class It has 2 variables and 10 methods

24.2.2 Constructor & Destructor Documentation

IPAddress::IPAddress ( )

IPAddress::IPAddress (std::string x)

IPAddress::IPAddress (const **IPAddress** & x)

IPAddress::IPAddress (smart_ptr< **IPAddress** > x)

IPAddress::~IPAddress ( ) [virtual]

24.2.3 Member Function Documentation

int32_t IPAddress::compare (const os::smart_ptr< **IPAddress** > comp) const

Compares IP Addresses Compares two IPAddresses and returns the difference.

Returns

int32_t

int32_t IPAddress::compare (const **IPAddress** * comp) const

const char* os::IPAddress::getConstAddress ( ) const [inline]

Return IP Address Gives a pointer to the **IPAddress** (p. 497) to the caller.

Returns

char*

bool os::IPAddress::isIPv6 ( ) const [inline]

Return if an IPAdress is IPv6 returns the _isIPv6 boolean to the caller.

Returns

bool

char* os::IPAddress::printAddress ( ) [inline]

Return IP Address Gives a pointer to the **IPAddress** (p. 497) to the caller.

Returns

char*

24.2.4 Member Data Documentation

bool os::IPAddress::_isIPv6 [protected]

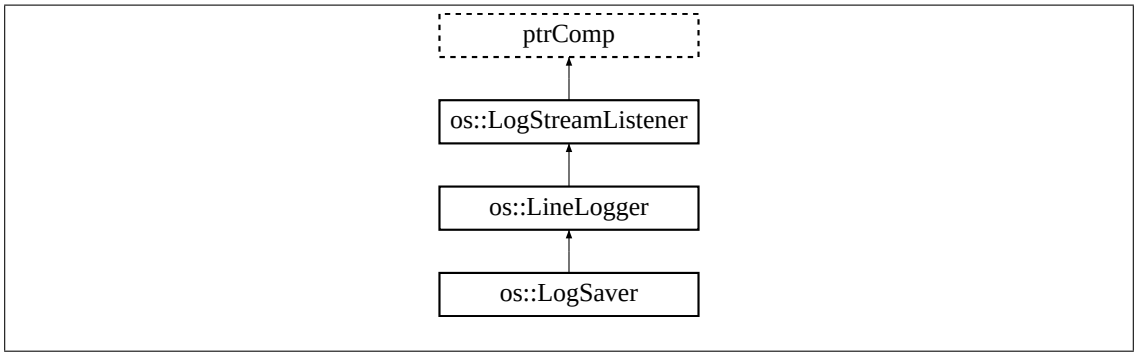
Holds if an address is IPv6 An address has this flag for easy analysis of IPv6 and IPv4 status. Thanks to this, we can use one IP address class for both standards.

char os::IPAddress::name[80] [protected]

Actual data of IP address. The actual IP is a critical element, as it allows for the core communication. We allocate 80 bytes so that we have enough memory for both the IPv4 and IPv6 addresses.

24.3 os::LineLogger Class Reference

Inheritance diagram for os::LineLogger:



Public Member Functions

- **LineLogger** (smart_ptr< std::ostream > s)
- **LineLogger** (smart_ptr< std::ostream > s, std::string head)
- virtual ~**LineLogger** ()
- void **setLogTime** (bool lt)
- void **setPrintName** (bool pn)
- void **setIsOn** (bool io)
- bool **getLogTime** () const
- bool **getPrintName** () const
- bool **isOn** () const
- void **setSpecialCase** (bool isSpecial)
- bool **getSpecialCase** () const
- virtual void **logHeader** (uint64_t timestamp, **LogDirectedStream** &source)
- virtual void **logChar** (int32_t c)
- virtual void **log_endl** ()
- virtual void **log** (smart_ptr< **logLine** > line)
- virtual void **emergencyNewLine** (**LogDirectedStream** &source)
- virtual void **emergencyArrival** (**LogDirectedStream** &source, int32_t c)
- virtual void **stringArrival** (smart_ptr< **logLine** > line)

Protected Attributes

- bool **isSpecialCase**
- smart_ptr< std::ostream > **strm**
- bool **_isOn**
- bool **logTime**
- bool **printName**

24.3.1 Constructor & Destructor Documentation

LineLogger::LineLogger (smart_ptr< std::ostream > s)

LineLogger::LineLogger (smart_ptr< std::ostream > s, std::string head)

virtual os::LineLogger::~LineLogger ( ) [inline], [virtual]

24.3.2 Member Function Documentation

void LineLogger::emergencyArrival (**LogDirectedStream** & source, int32_t c) [virtual]

Reimplemented from **os::LogStreamListener** (p. 514).

void LineLogger::emergencyNewLine (**LogDirectedStream** & source) [virtual]

Reimplemented from **os::LogStreamListener** (p. 514).

bool os::LineLogger::getLogTime ( ) const [inline]

bool os::LineLogger::getPrintName ( ) const [inline]

bool os::LineLogger::getSpecialCase ( ) const [inline]

bool os::LineLogger::isOn ( ) const [inline]

void LineLogger::log (smart_ptr< **logLine** > line) [virtual]

Reimplemented in **os::LogSaver** (p. 510).

void LineLogger::log_endl ( ) [virtual]

Reimplemented in **os::LogSaver** (p. 510).

void LineLogger::logChar (int32_t c) [virtual]

Reimplemented in **os::LogSaver** (p. 510).

void LineLogger::logHeader (uint64_t timestamp, **LogDirectedStream** & source) [virtual]

Reimplemented in **os::LogSaver** (p. 510).

void os::LineLogger::setIsOn (bool io) [inline]

void os::LineLogger::setLogTime (bool lt) [inline]

void os::LineLogger::setPrintName (bool pn) [inline]

void os::LineLogger::setSpecialCase (bool isSpecial) [inline]

void LineLogger::stringArrival (smart_ptr< **logLine** > line) [virtual]

Reimplemented from **os::LogStreamListener** (p. 514).

24.3.3 Member Data Documentation

bool os::LineLogger::_isOn [protected]

bool os::LineLogger::isSpecialCase [protected]

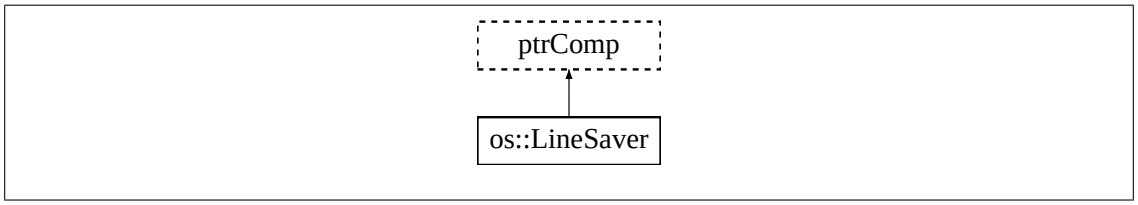
bool os::LineLogger::logTime [protected]

bool os::LineLogger::printName [protected]

smart_ptr<std::ostream> os::LineLogger::strm [protected]

24.4 os::LineSaver Class Reference

Inheritance diagram for os::LineSaver:



Public Member Functions

- **LineSaver** ()
- virtual ~**LineSaver** ()
- void **setSize** (uint32_t s)
- uint32_t **getSize** () const
- uint32_t **getNumLines** () const
- void **addListener** (smart_ptr< **LineSaverListener** > l)
- void **removeListener** (smart_ptr< **LineSaverListener** > l)
- void **pushLine** (smart_ptr< **logLine** > line)
- smart_ptr< **logLine** > **popLine** (uint32_t x)
- void **setSpecialCase** (bool isSpecial)
- bool **getSpecialCase** () const

Private Member Functions

- void **priv_addListener** (smart_ptr< **LineSaverListener** > l)
- void **priv_removeListener** (smart_ptr< **LineSaverListener** > l)

Private Attributes

- bool **isSpecialCase**
- uint32_t **size**
- uint32_t **beginning_pos**
- uint32_t **next_pos**
- std::vector< smart_ptr< **logLine** > > **lineList**

- **os::spinLock lock**
- smartSet< **LineSaverListener** > **listener**

Friends

- class **LineSaverListener**

24.4.1 Constructor & Destructor Documentation

LineSaver::LineSaver ( )

LineSaver::~LineSaver ( ) [virtual]

24.4.2 Member Function Documentation

void LineSaver::addListener (smart_ptr< **LineSaverListener** > l)

uint32_t LineSaver::getNumLines ( ) const

uint32_t os::LineSaver::getSize ( ) const [inline]

bool os::LineSaver::getSpecialCase ( ) const [inline]

smart_ptr< **logLine** > LineSaver::popLine (uint32_t x)

void LineSaver::priv_addListener (smart_ptr< **LineSaverListener** > l) [private]

void LineSaver::priv_removeListener (smart_ptr< **LineSaverListener** > l) [private]

void LineSaver::pushLine (smart_ptr< **logLine** > line)

void LineSaver::removeListener (smart_ptr< **LineSaverListener** > l)

void LineSaver::setSize (uint32_t s)

void os::LineSaver::setSpecialCase (bool isSpecial) [inline]

24.4.3 Friends And Related Function Documentation

friend class **LineSaverListener** [friend]

24.4.4 Member Data Documentation

uint32_t os::LineSaver::beginning_pos [private]

bool os::LineSaver::isSpecialCase [private]

std::vector<smart_ptr<**logLine**> > os::LineSaver::lineList [private]

smartSet<**LineSaverListener**> os::LineSaver::listener [private]

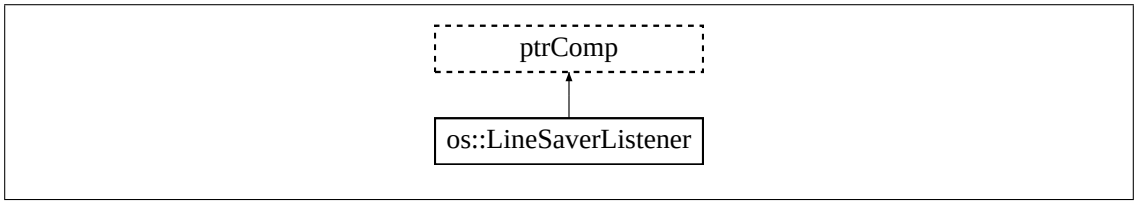
os::spinLock os::LineSaver::lock [private]

uint32_t os::LineSaver::next_pos [private]

uint32_t os::LineSaver::size [private]

24.5 os::LineSaverListener Class Reference

Inheritance diagram for os::LineSaverListener:



Public Member Functions

- virtual ~**LineSaverListener** ()
- void **addSaver** (smart_ptr< **LineSaver** > l)
- void **removeSaver** (smart_ptr< **LineSaver** > l)

Protected Member Functions

- virtual void **receiveLine** (smart_ptr< **LineSaver** > source, smart_ptr< **logLine** > message)

Private Member Functions

- void **priv_addSaver** (smart_ptr< **LineSaver** > l)
- void **priv_removeSaver** (smart_ptr< **LineSaver** > l)

Private Attributes

- **os::spinLock lock**
- smartSet< **LineSaver** > **saver**

Friends

- class **LineSaver**

24.5.1 Constructor & Destructor Documentation

LineSaverListener::~LineSaverListener ( ) `[virtual]`

24.5.2 Member Function Documentation

void LineSaverListener::addSaver (smart_ptr< **LineSaver** > l)

void LineSaverListener::priv_addSaver (smart_ptr< **LineSaver** > l) `[private]`

void LineSaverListener::priv_removeSaver (smart_ptr< **LineSaver** > l) `[private]`

virtual void os::LineSaverListener::receiveLine (smart_ptr< **LineSaver** > source, smart_ptr< **logLine** > message) `[inline]`, `[protected]`, `[virtual]`

void LineSaverListener::removeSaver (smart_ptr< **LineSaver** > l)

24.5.3 Friends And Related Function Documentation

friend class **LineSaver** `[friend]`

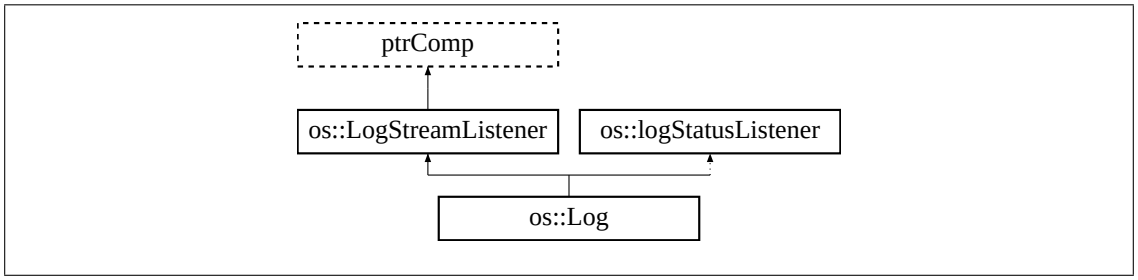
24.5.4 Member Data Documentation

os::spinLock os::LineSaverListener::lock `[private]`

smartSet<**LineSaver**> os::LineSaverListener::saver `[private]`

24.6 os::Log Class Reference

Inheritance diagram for os::Log:



Public Member Functions

- virtual ~**Log** ()
- void **pushFilePath** (std::string fp)
- void **startSavingThread** ()
- void **savingThread** (os::smart_ptr< **os::threadHolder** > th)
- std::string **getFilePath** () const
- bool **isSavingThreadActive** () const
- void **exitSaveThread** ()
- const **LineSaver** & **getMainStream** () const
- smart_ptr< **LineSaver** > **getMainStreamReference** ()
- void **setSaverSize** (uint32_t size)
- void **setSaveIsOn** (bool ison)
- void **setSaveLogTime** (bool lt)
- void **setSaveLogName** (bool ln)
- void **setSTDIsOn** (bool ison)
- void **setSTDLogTime** (bool lt)
- void **setSTDLogName** (bool ln)
- void **setIsOn** (bool ison)
- void **setLogTime** (bool lt)
- void **setLogName** (bool ln)
- smart_ptr< **LogDirectedStream** > **newTargetStream** (std::string name, bool isError)
- smart_ptr< std::ostream > **castNewTargetStream** (std::string name, bool isError)

- smart_ptr< **LogDirectedStream** > **find** (std::string name)
- smart_ptr< std::ostream > **castFind** (std::string name)
- bool **deleteStream** (std::string name)
- virtual void **stringArrival** (smart_ptr< **logLine** > line)
- void **removeListener** (smart_ptr< **LogStreamListener** > lst)

Static Public Member Functions

- static smart_ptr< **Log** > **singleton** ()

Protected Member Functions

- void **receiveChanged** (smart_ptr< **logStatusHolder** > h)

Private Member Functions

- **Log** ()

Private Attributes

- AVLTree< **LogDirectedStream** > **streams**
- smart_ptr< **LineLogger** > **std_log**
- smart_ptr< **LineLogger** > **err_log**
- **spinLock mainStreamLock**
- **LineSaver mainStream**
- std::string **file_path**
- smart_ptr< **LogSaver** > **primary_save_file**
- smart_ptr< **LogSaver** > **std_save_file**
- smart_ptr< **LogSaver** > **err_save_file**
- **spinLock logSaverLock**
- bool **isSavingThread**
- bool **continue_saving**

24.6.1 Constructor & Destructor Documentation

Log::Log ( ) `[private]`

Log::~Log ( ) `[virtual]`

24.6.2 Member Function Documentation

smart_ptr< std::ostream > Log::castFind (std::string name)

smart_ptr< std::ostream > Log::castNewTargetStream (std::string name, bool isError)

bool Log::deleteStream (std::string name)

void os::Log::exitSaveThread ( ) `[inline]`

smart_ptr< **LogDirectedStream** > Log::find (std::string name)

std::string os::Log::getFilePath ( ) const [inline]

const **LineSaver**& os::Log::getMainStream ( ) const [inline]

smart_ptr<**LineSaver**> os::Log::getMainStreamReference ( ) [inline]

bool os::Log::isSavingThreadActive ( ) const [inline]

smart_ptr< **LogDirectedStream** > Log::newTargetStream (std::string name, bool isError)

void Log::pushFilePath (std::string fp)

void Log::receiveChanged (smart_ptr< **logStatusHolder** > h) [protected], [virtual]

Reimplemented from **os::logStatusListener** (p. 513).

void Log::removeListener (smart_ptr< **LogStreamListener** > lst)

void Log::savingThread (os::smart_ptr< **os::threadHolder** > th)

void Log::setIsOn (bool ison)

void Log::setLogName (bool ln)

void Log::setLogTime (bool lt)

void Log::setSaveIsOn (bool ison)

void Log::setSaveLogName (bool ln)

void Log::setSaveLogTime (bool lt)

void Log::setSaverSize (uint32_t size)

void Log::setSTDIsOn (bool ison)

void Log::setSTDLogName (bool ln)

void Log::setSTDLogTime (bool lt)

smart_ptr< **Log** > Log::singleton ( ) [static]

void Log::startSavingThread ( )

void Log::stringArrival (smart_ptr< **logLine** > line) [virtual]

Reimplemented from **os::LogStreamListener** (p. 514).

24.6.3 Member Data Documentation

bool os::Log::continue_saving [private]

smart_ptr<**LineLogger**> os::Log::err_log [private]

smart_ptr<**LogSaver**> os::Log::err_save_file [private]

std::string os::Log::file_path [private]

bool os::Log::isSavingThread [private]

spinLock os::Log::logSaverLock [private]

LineSaver os::Log::mainStream [private]

spinLock os::Log::mainStreamLock [private]

smart_ptr<**LogSaver**> os::Log::primary_save_file [private]

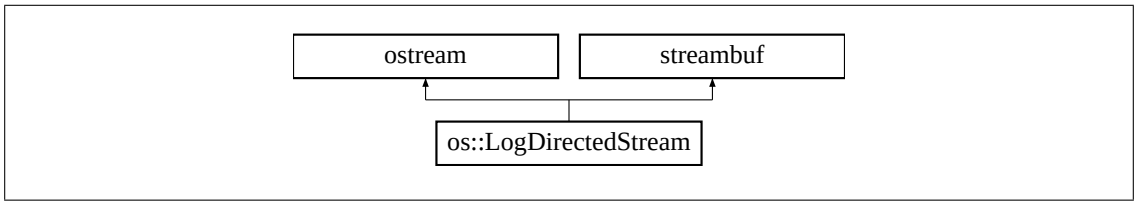
smart_ptr<**LineLogger**> os::Log::std_log [private]

smart_ptr<**LogSaver**> os::Log::std_save_file [private]

AVLTree<**LogDirectedStream**> os::Log::streams [private]

24.7 os::LogDirectedStream Class Reference

Inheritance diagram for os::LogDirectedStream:



Public Member Functions

- **LogDirectedStream** (std::string n, bool ie)
- virtual ~**LogDirectedStream** ()
- virtual int32_t **overflow** (int32_t c)
- void **addListener** (smart_ptr< **LogStreamListener** > lst)
- void **removeListener** (smart_ptr< **LogStreamListener** > lst)
- std::string **getName** () const
- bool **isError** () const
- const **LineSaver** & **getLines** () const
- **LineSaver** & **modLines** ()
- const bool **operator==** (const **LogDirectedStream** &comp) const
- const bool **operator**> (const **LogDirectedStream** &comp) const

Private Attributes

- std::string **name**
- bool **_isError**
- **LineSaver saver**
- smartSet< **LogStreamListener** > **listeners**
- std::string **current_line**

24.7.1 Constructor & Destructor Documentation

LogDirectedStream::LogDirectedStream (std::string n, bool ie)

virtual os::LogDirectedStream::~LogDirectedStream ( ) [inline], [virtual]

24.7.2 Member Function Documentation

void LogDirectedStream::addListener (smart_ptr< **LogStreamListener** > lst)

const **LineSaver**& os::LogDirectedStream::getLines ( ) const [inline]

std::string os::LogDirectedStream::getName ( ) const [inline]

bool os::LogDirectedStream::isError ( ) const [inline]

LineSaver& os::LogDirectedStream::modLines ( ) [inline]

const bool LogDirectedStream::operator== (const **LogDirectedStream** & comp) const

const bool LogDirectedStream::operator> (const **LogDirectedStream** & comp) const

int32_t LogDirectedStream::overflow (int32_t c) [virtual]

void LogDirectedStream::removeListener (smart_ptr< **LogStreamListener** > lst)

24.7.3 Member Data Documentation

bool os::LogDirectedStream::_isError [private]

std::string os::LogDirectedStream::current_line [private]

smartSet<**LogStreamListener**> os::LogDirectedStream::listeners [private]

std::string os::LogDirectedStream::name [private]

LineSaver os::LogDirectedStream::saver [private]

24.8 os::logLine Struct Reference

Public Member Functions

- virtual ~**logLine** ()

Public Attributes

- smart_ptr< **LogDirectedStream** > **source**
- std::string **line**
- uint64_t **timestamp**

24.8.1 Constructor & Destructor Documentation

virtual os::logLine::~logLine ( ) [inline], [virtual]

24.8.2 Member Data Documentation

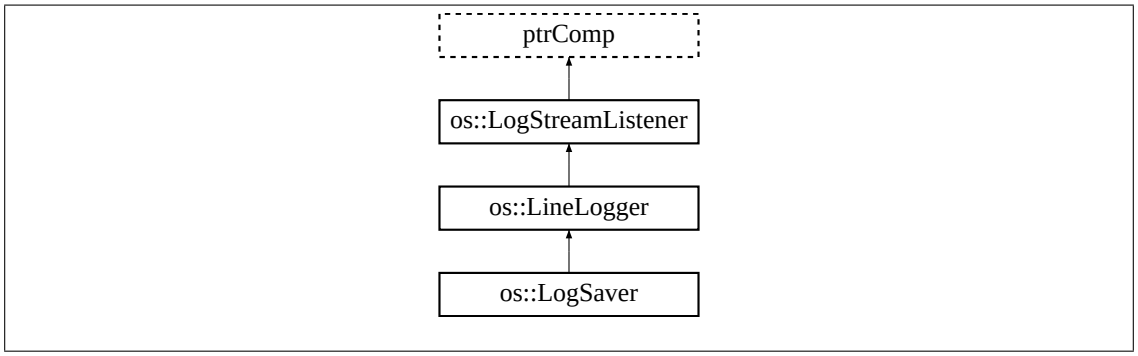
std::string os::logLine::line

smart_ptr<**LogDirectedStream**> os::logLine::source

uint64_t os::logLine::timestamp

24.9 os::LogSaver Class Reference

Inheritance diagram for os::LogSaver:



Public Member Functions

- **LogSaver** (std::string file)
- **LogSaver** (std::string file, std::string head)
- virtual ~**LogSaver** ()
- void **saveLog** ()
- virtual void **logHeader** (uint64_t timestamp, **LogDirectedStream** &source)
- virtual void **logChar** (int32_t c)
- virtual void **log_endl** ()
- virtual void **log** (smart_ptr< **logLine** > line)

Private Attributes

- **os::spinLock file_lock**
- std::queue< smart_ptr< **logLine** > > **lineQueue**

Additional Inherited Members

24.9.1 Constructor & Destructor Documentation

LogSaver::LogSaver (std::string file)

LogSaver::LogSaver (std::string file, std::string head)

LogSaver::~LogSaver ( ) `[virtual]`

24.9.2 Member Function Documentation

void LogSaver::log (smart_ptr< **logLine** > line) [virtual]

Reimplemented from **os::LineLogger** (p. 500).

void LogSaver::log_endl ( ) [virtual]

Reimplemented from **os::LineLogger** (p. 500).

void LogSaver::logChar (int32_t c) [virtual]

Reimplemented from **os::LineLogger** (p. 500).

void LogSaver::logHeader (uint64_t timestamp, **LogDirectedStream** & source) [virtual]

Reimplemented from **os::LineLogger** (p. 500).

void LogSaver::saveLog ( )

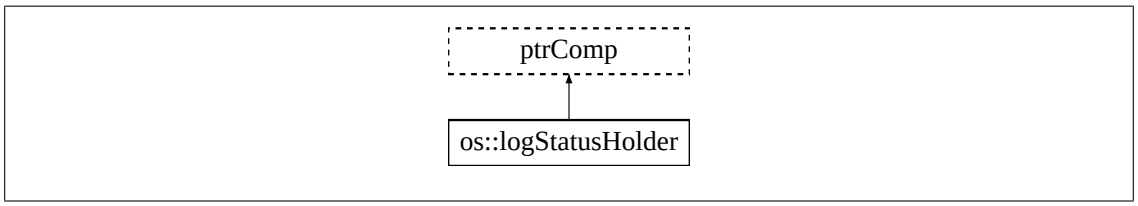
24.9.3 Member Data Documentation

os::spinLock os::LogSaver::file_lock [private]

std::queue<smart_ptr<**logLine**> > os::LogSaver::lineQueue [private]

24.10 os::logStatusHolder Class Reference

Inheritance diagram for os::logStatusHolder:



Public Member Functions

- **logStatusHolder** ()
- virtual ~**logStatusHolder** ()
- void **addListener** (smart_ptr< **logStatusListener** > l)
- void **removeListener** (smart_ptr< **logStatusListener** > l)
- void **setSTDStatus** (bool std)
- void **setERRStatus** (bool err)
- bool **getSTDStatus** () const
- bool **getERRStatus** () const

Private Member Functions

- void **triggerChange** ()
- void **priv_addListener** (smart_ptr< **logStatusListener** > l)
- void **priv_removeListener** (smart_ptr< **logStatusListener** > l)

Private Attributes

- bool **stdstatus**
- bool **errstatus**
- **spinLock lisLock**
- smartSet< **logStatusListener** > **listener**

Friends

- class **logStatusListener**

24.10.1 Constructor & Destructor Documentation

logStatusHolder::logStatusHolder ( )

logStatusHolder::~logStatusHolder ( ) [virtual]

24.10.2 Member Function Documentation

void logStatusHolder::addListener (smart_ptr< **logStatusListener** > l)

bool os::logStatusHolder::getERRStatus ( ) const [inline]

bool os::logStatusHolder::getSTDStatus ( ) const [inline]

void logStatusHolder::priv_addListener (smart_ptr< **logStatusListener** > l) [private]

void logStatusHolder::priv_removeListener (smart_ptr< **logStatusListener** > l) [private]

void logStatusHolder::removeListener (smart_ptr< **logStatusListener** > l)

void logStatusHolder::setERRStatus (bool err)

void logStatusHolder::setSTDStatus (bool std)

void logStatusHolder::triggerChange ( ) [private]

24.10.3 Friends And Related Function Documentation

friend class **logStatusListener** [friend]

24.10.4 Member Data Documentation

bool os::logStatusHolder::errstatus [private]

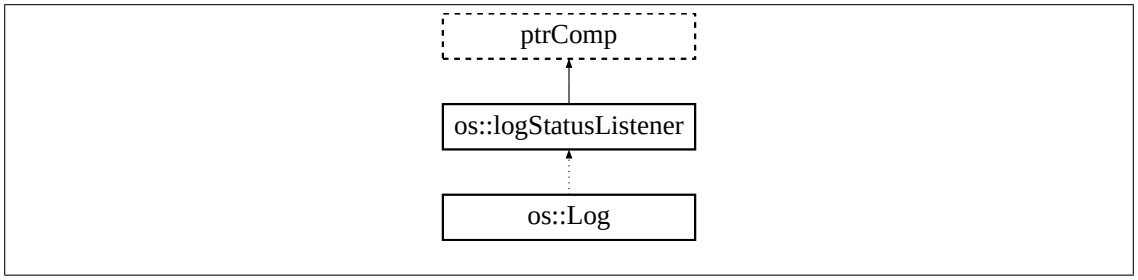
spinLock os::logStatusHolder::lisLock [private]

smartSet<**logStatusListener**> os::logStatusHolder::listener [private]

bool os::logStatusHolder::stdstatus [private]

24.11 os::logStatusListener Class Reference

Inheritance diagram for os::logStatusListener:



Public Member Functions

- virtual ~**logStatusListener** ()
- void **addHolder** (smart_ptr< **logStatusHolder** > l)
- void **removeHolder** (smart_ptr< **logStatusHolder** > l)

Protected Member Functions

- virtual void **receiveChanged** (smart_ptr< **logStatusHolder** > h)

Private Member Functions

- void **priv_addHolder** (smart_ptr< **logStatusHolder** > l)
- void **priv_removeHolder** (smart_ptr< **logStatusHolder** > l)

Private Attributes

- **spinLock senLock**
- smartSet< **logStatusHolder** > **sender**

Friends

- class **logStatusHolder**

24.11.1 Constructor & Destructor Documentation

logStatusListener::~logStatusListener ( ) [virtual]

24.11.2 Member Function Documentation

void logStatusListener::addHolder (smart_ptr< **logStatusHolder** > l)

void logStatusListener::priv_addHolder (smart_ptr< **logStatusHolder** > l) [private]

void logStatusListener::priv_removeHolder (smart_ptr< **logStatusHolder** > l) [private]

virtual void os::logStatusListener::receiveChanged (smart_ptr< **logStatusHolder** > h) [inline], [protected], [virtual]

Reimplemented in **os::Log** (p. 506).

void logStatusListener::removeHolder (smart_ptr< **logStatusHolder** > l)

24.11.3 Friends And Related Function Documentation

friend class **logStatusHolder** [friend]

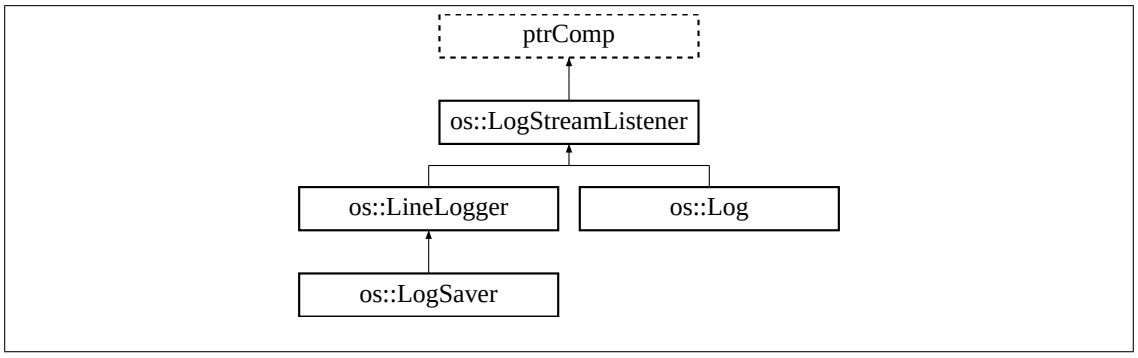
24.11.4 Member Data Documentation

smartSet<**logStatusHolder**> os::logStatusListener::sender [private]

spinLock os::logStatusListener::senLock [private]

24.12 os::LogStreamListener Class Reference

Inheritance diagram for os::LogStreamListener:



Public Member Functions

- virtual ~**LogStreamListener** ()
- virtual void **emergencyNewLine** (**LogDirectedStream** &source)
- virtual void **emergencyArrival** (**LogDirectedStream** &source, int32_t c)
- virtual void **stringArrival** (smart_ptr< **logLine** > line)

24.12.1 Constructor & Destructor Documentation

virtual os::LogStreamListener::~LogStreamListener ( ) [inline], [virtual]

24.12.2 Member Function Documentation

virtual void os::LogStreamListener::emergencyArrival (**LogDirectedStream** & source, int32_t c) [inline], [virtual]

Reimplemented in **os::LineLogger** (p. 500).

virtual void os::LogStreamListener::emergencyNewLine (**LogDirectedStream** & source) [inline], [virtual]

Reimplemented in **os::LineLogger** (p. 500).

virtual void os::LogStreamListener::stringArrival (smart_ptr< **logLine** > line) [inline], [virtual]

Reimplemented in **os::Log** (p. 506), and **os::LineLogger** (p. 500).

24.13 os::multiLock Class Reference

os::multilock class definition Defines the os::multilock class. This class has 4 variables and 8 methods

Public Member Functions

- **multiLock** ()

  multilock constructor This method constructs a multilock. We do not have a copy constructor or any custom constructors of any type.
- virtual ~**multiLock** ()

  multilock destructor The destructor is virtual, so that if we decide to inherent for any reason, the multilock class will already be prepared.
- bool **isLocked** () const

  returns _locked status This allows us to determine whether or not a multilock is being written to. This can be used both in general classes as well as in the multilock class as well.
- uint32_t **getCounter** () const

  returns _counter The amount of readers is variable, so this method returns the amount of readers currently looking at the multilock. This method can be used both by those outside the multilock as well as by other multilock methods.
- void **increment** ()

  allows someone to read When a user wants to read, they can call increment. This ensures that the multilock can be safely read. If the multilock cannot be read, increment waits until a read is available.
- void **decrement** ()

  allows a thread to stop reading the multilock When a thread has finished reading, it calls decrement in order to return the multilock. This ensures that other threads can read the multilock
- void **lock** ()

  allows a thread to write Only one thread can write at any given time. The lock method ensures that no one else is reading or writing, then acquires and writes.
- void **unlock** ()

  allows other threads to write Once a thread has finished writing, it should call unlock so that other threads can read and write. The unlock method ends the period of writing for a thread.

Private Attributes

- **spinLock dLock**

  spinlock The spinlock prevents a multilock from being modified by more than 1 thread at any given time. If more than 1 thread were to attempt to read for example, a miscount of the number of readers could permanently lock the multilock or allow for unsafe writing.

- bool **_locked**

  locked boolean The _locked variable is to indicate whether or nor a multilock is being written. Should a multilock be actively written, it would be unsafe to read or have another thread attempt to read.

- uint32_t **_counter**

  counter for readers The _counter represents how many threads are currently reading though whatever the multilock is protecting.

- uint32_t **_max**

  most allowable readers Given that there may be a maximum amount of threads that should be reading something at any given time, the _max variable allows us to define how many readers a given multilock can have.

24.13.1 Detailed Description

os::multilock class definition Defines the os::multilock class. This class has 4 variables and 8 methods

24.13.2 Constructor & Destructor Documentation

multiLock::multiLock ( )

multilock constructor This method constructs a multilock. We do not have a copy constructor or any custom constructors of any type.

multiLock::~multiLock ( ) `[virtual]`

multilock destructor The destructor is virtual, so that if we decide to inherent for any reason, the multilock class will already be prepared.

24.13.3 Member Function Documentation

void multiLock::decrement ( )

allows a thread to stop reading the multilock When a thread has finished reading, it calls decrement in order to return the multilock. This ensures that other threads can read the multilock

Returns

void

uint32_t os::multiLock::getCounter ( ) const `[inline]`

returns _counter The amount of readers is variable, so this method returns the amount of readers currently looking at the multilock. This method can be used both by those outside the multilock as well as by other multilock methods.

Returns

uint32_t

void multiLock::increment ( )

allows someone to read When a user wants to read, they can call increment. This ensures that the multilock can be safely read. If the multilock cannot be read, increment waits until a read is available.

Returns

void

bool os::multiLock::isLocked ( ) const [inline]

returns _locked status This allows us to determine whether or not a multilock is being written to. This can be used both in general classes as well as in the multilock class as well.

Returns

boolean

void multiLock::lock ( )

allows a thread to write Only one thread can write at any given time. The lock method ensures that no one else is reading or writing, then acquires and writes.

Returns

void

void multiLock::unlock ( )

allows other threads to write Once a thread has finished writing, it should call unlock so that other threads can read and write. The unlock method ends the period of writing for a thread.

Returns

void

24.13.4 Member Data Documentation

uint32_t os::multiLock::_counter [private]

counter for readers The _counter represents how many threads are currently reading though whatever the multilock is protecting.

bool os::multiLock::_locked [private]

locked boolean The _locked variable is to indicate whether or nor a multilock is being written. Should a multilock be actively written, it would be unsafe to read or have another thread attempt to read.

uint32_t os::multiLock::_max [private]

most allowable readers Given that there may be a maximum amount of threads that should be reading something at any given time, the _max variable allows us to define how many readers a given multilock can have.

spinLock os::multiLock::dLock [private]

spinlock The spinlock prevents a multilock from being modified by more than 1 thread at any given time. If more than 1 thread were to attempt to read for example, a miscount of the number of readers could permanently lock the multilock or allow for unsafe writing.

24.14 os::myIPAddress Class Reference

Holds a node's own IP address Every node needs it's own IP address. This class holds that value, as well as provide several functions for determining priorities.

Public Member Functions

- **myIPAddress** ()
- virtual ~**myIPAddress** ()
- **IPAddress getAddress** ()
- char * **getIPString** ()
- **IPAddress getIPv6Address** ()
- **IPAddress getIPv4Address** ()
- bool **isMe** (const **IPAddress** &addr) const
- bool **isV6Priority** () const
- void **setV6Priority** (bool p)

Private Member Functions

- **IPAddress resetAddress** ()

Private Attributes

- **IPAddress address**
- **IPAddress ip6Address**
- **IPAddress ip4Address**
- clock_t **last**
- bool **v6_prioriity**

24.14.1 Detailed Description

Holds a node's own IP address Every node needs it's own IP address. This class holds that value, as well as provide several functions for determining priorities.

24.14.2 Constructor & Destructor Documentation

myIPAddress::myIPAddress ( )

myIPAddress::~myIPAddress ( ) [virtual]

24.14.3 Member Function Documentation

IPAddress myIPAddress::getAddress ( )

brief returns address Updates the IP addresses if necessary and return the current IP address.

Returns

IPAddress (p. 497)

char * myIPAddress::getIPString ( )

brief Returns IP address as a string Returns the IP address as a string.

Returns

char*

IPAddress os::myIPAddress::getIPv4Address ( ) [inline]

brief gives IPv4 Returns the IPv4 address.

Returns

IPAddress (p. 497)

IPAddress os::myIPAddress::getIPv6Address ( ) [inline]

brief gives IPv6 Returns the IPv6 address.

Returns

IPAddress (p. 497)

bool myIPAddress::isMe (const **IPAddress** & addr) const

brief is a given address mine Determines if a given address is the same as the address of this object.

Returns

bool

bool os::myIPAddress::isV6Priority ( ) const [inline]

brief returns IPv6 priority Tells the caller if this object prioritizes IPv6.

Returns

bool

IPAddress myIPAddress::resetAddress ( ) [private]

brief resets IP address Resets the IP addresses of the object.

Returns

IPAddress (p. 497)

void myIPAddress::setV6Priority (bool p)

brief sets IPv6 priority Sets IPv6 priority.

24.14.4 Member Data Documentation

IPAddress os::myIPAddress::address [private]

brief IP address A base IP address, can be either IPv4 or IPv6.

IPAddress os::myIPAddress::ip4Address [private]

brief IPv4 address The IPv4 address of the current object.

IPAddress os::myIPAddress::ip6Address [private]

brief IPv6 address The IPv6 address of the current object.

clock_t os::myIPAddress::last [private]

brief last time IP address updated Holds the last time the IP addresses were updated.

bool os::myIPAddress::v6_prioriity [private]

brief IP type priority Indicates if a node wants to prioritize IPv6.

24.15 os::safeQueue< dataType > Class Template Reference

This is the **safeQueue** (p. 519) class The **safeQueue** (p. 519) class is thread safe. It is a template class.

Public Member Functions

- **safeQueue** (int32_t s)
- **safeQueue** ()
- virtual ~**safeQueue** ()
- void **push** (smart_ptr< dataType > x)

  Add a value to the queue This method pushes an element into the queue, assuming that space exists in the queue.
- smart_ptr< dataType > **pop** ()

Removes the head of the queue This method pops the head of the queue, if it exists. It also adjusts the queue so that the next element is available without moving every element in the queue.

- bool **empty** ()

  Checks if the queue is empty. This method checks if any elements are in the queue.

Private Attributes

- int32_t **size**

  Size of the queue We store the size of the queue. This integer holds that value.
- smart_ptr< dataType > ∗ **array**

  Elements in the queue We store all the elements for the queue is this vector.
- int32_t **start**

  The start of the queue This stores the location of the start of the queue.
- int32_t **end**

  The end of the queue This stores the end of the queue.
- **spinLock lock**

  *Lock for threads This **spinLock** (p. 537) allows us to safely access the queue from multiple threads.*

24.15.1 Detailed Description

template<class dataType>
class os::safeQueue< dataType >

This is the **safeQueue** (p. 519) class The **safeQueue** (p. 519) class is thread safe. It is a template class.

24.15.2 Constructor & Destructor Documentation

template<class dataType> **os::safeQueue**< dataType >::**safeQueue** (int32_t s) `[inline]`

template<class dataType> **os::safeQueue**< dataType >::**safeQueue** ( ) `[inline]`

template<class dataType> virtual **os::safeQueue**< dataType >::~**safeQueue** ( ) `[inline]`, `[virtual]`

24.15.3 Member Function Documentation

template<class dataType> bool **os::safeQueue**< dataType >::empty ( ) `[inline]`

Checks if the queue is empty. This method checks if any elements are in the queue.

Returns

bool

template<class dataType> smart_ptr<dataType> **os::safeQueue**< dataType >::pop ( ) [inline]

Removes the head of the queue This method pops the head of the queue, if it exists. It also adjusts the queue so that the next element is available without moving every element in the queue.

Returns

smart_ptr<dataType>

template<class dataType> void **os::safeQueue**< dataType >::push (smart_ptr< dataType > x) [inline]

Add a value to the queue This method pushes an element into the queue, assuming that space exists in the queue.

24.15.4 Member Data Documentation

template<class dataType> smart_ptr<dataType>* **os::safeQueue**< dataType >::array [private]

Elements in the queue We store all the elements for the queue is this vector.

template<class dataType> int32_t **os::safeQueue**< dataType >::end [private]

The end of the queue This stores the end of the queue.

template<class dataType> **spinLock os::safeQueue**< dataType >::lock [private]

Lock for threads This **spinLock** (p. 537) allows us to safely access the queue from multiple threads.

template<class dataType> int32_t **os::safeQueue**< dataType >::size [private]

Size of the queue We store the size of the queue. This integer holds that value.

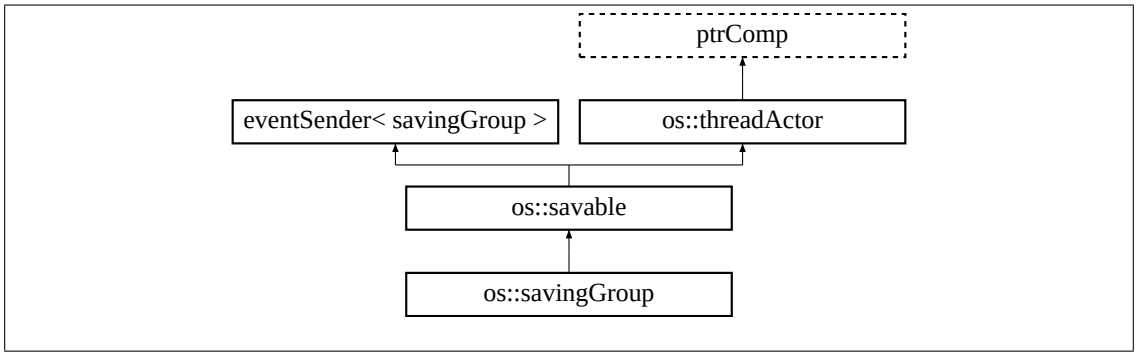
template<class dataType> int32_t **os::safeQueue**< dataType >::start [private]

The start of the queue This stores the location of the start of the queue.

24.16 os::savable Class Reference

Basic saving class.

Inheritance diagram for os::savable:



Public Member Functions

- **savable** ()

  Default savable constructor.
- virtual ~**savable** ()

  Virtual destructor.
- void **markChanged** ()

  Mark change in savable class.
- bool **needsSaving** () const

  Check if class needs to be saved.
- virtual void **save** ()

  Saves the class.
- bool **hasError** () const

  Returns if this class has a logged error.
- const std::string & **getLastError** () const

  Returns the logged error.
- void **clearError** ()

  Clears the logged error.
- void **perform_action** ()

  Saves the class.
- virtual bool **singleCase** () const

  Indicates if a thread is a single function.
- virtual bool **action_ready** ()

  Indicates if a function is available.

Static Public Member Functions

- static smart_ptr< **threadDistributor** > **getThread** ()

  Access saving thread.
- static void **unbindThread** ()

  Un-bind saving thread.
- static bool **setThread** (smart_ptr< **threadDistributor** > thr)

  Set the saving thread-distributor.

Protected Member Functions

- void **finishedSaving** ()

  *Reset **os::savable::_needsSaving*** *(p. 526).*
- void **errorSaving** (std::string err)

  Logs error while saving.

Private Attributes

- bool **_needsSaving**

  Holds if this class needs saving.
- std::string **lastError**

  String representation of the last error.

24.16.1 Detailed Description

Basic saving class.

Class which defines itself as savable. A savable class alerts its listeners when it needs to be re-saved.

24.16.2 Constructor & Destructor Documentation

os::savable::savable ( )

Default savable constructor.

virtual os::savable::~savable ( ) [inline], [virtual]

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

24.16.3 Member Function Documentation

virtual bool os::savable::action_ready ( ) [inline], [virtual]

Indicates if a function is available.

Returns

needsSaving

Reimplemented from **os::threadActor** (p. 540).

void os::savable::clearError ( )

Clears the logged error.

Returns

void

void os::savable::errorSaving (std::string err) [protected]

Logs error while saving.
Sets the savable class into the error state, logging the given error.

Parameters

in	*err*	Error to be logged

Returns

void

void os::savable::finishedSaving ( ) [protected]

Reset **os::savable::_needsSaving** (p. 526).
Called when a savable class has finished saving itself.

Returns

void

const std::string& os::savable::getLastError ( ) const [inline]

Returns the logged error.

Returns

os::savable::_lastError

static smart_ptr<**threadDistributor**> os::savable::getThread ( ) [static]

Access saving thread.
Returns a reference to the thread-distributor used to save files. This distributor is NULL unless explicitly set.

Returns

Saving thread-distributor

bool os::savable::hasError ( ) const [inline]

Returns if this class has a logged error.

Returns

os::savable::_lastError != ""

void os::savable::markChanged ( )

Mark change in savable class.
Slots this class for saving. Flips **os::savable::_needsSaving** (p. 526).

Returns

void

bool os::savable::needsSaving ( ) const [inline]

Check if class needs to be saved.

Returns

os::savable::_needsSaving (p. 526)

void os::savable::perform_action ( ) [inline], [virtual]

Saves the class.

Returns

void

Reimplemented from **os::threadActor** (p. 540).

virtual void os::savable::save ( ) [inline], [virtual]

Saves the class.
This function must be re-implemented by classes which inherit from the savable class.

Returns

void

Reimplemented in **os::savingGroup** (p. 528).

static bool os::savable::setThread (smart_ptr< **threadDistributor** > thr) [static]

Set the saving thread-distributor.
Will not set the thread-distributor if the provided arguement is NULL or the current saving thread-distributor is defined.

Returns

True if successful, else, False

virtual bool os::savable::singleCase ( ) const [inline], [virtual]

Indicates if a thread is a single function.

Returns

true

Reimplemented from **os::threadActor** (p. 540).

static void os::savable::unbindThread ( ) [static]

Un-bind saving thread.

Sets the current saving thread to NULL. Assuming the saving distributor is not shared, this will delete the thread-distributor.

Returns

void

24.16.4 Member Data Documentation

bool os::savable::_needsSaving [private]

Holds if this class needs saving.

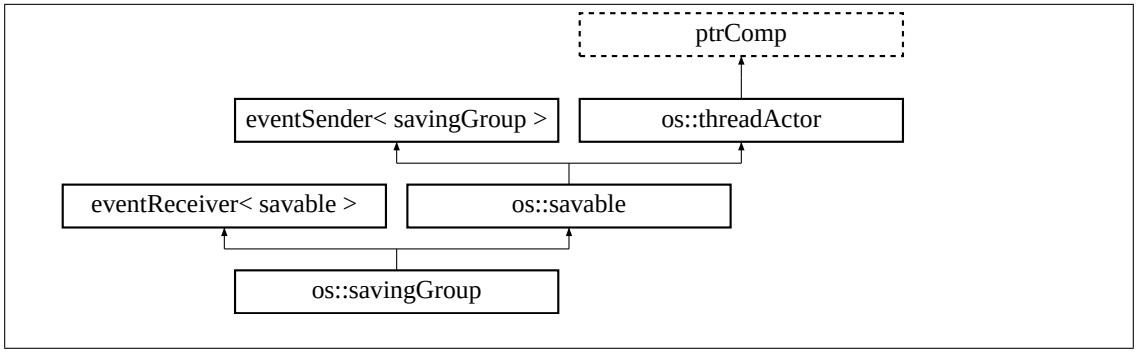
std::string os::savable::lastError [private]

String representation of the last error.

24.17 os::savingGroup Class Reference

Group of saving classes.

Inheritance diagram for os::savingGroup:



Public Member Functions

- **savingGroup** ()

  *Default **savingGroup** (p. 526) constructor.*
- virtual ~**savingGroup** ()

  Virtual destructor.
- virtual void **save** ()

  Saves the class.
- void **bindSavable** (smart_ptr< **savable** > sr)

  Checks if a savable class needs to be queued.

Protected Member Functions

- void **receiveEvent** (smart_ptr< **savable** > sr)

  Receives a change event.

Private Attributes

- **os::spinLock queueLock**

  Mutex for save queue.
- os::smart_ptr< os::unsortedList< **savable** > > **saveQueue**

  Queue of savable classes to be re-saved.

Friends

- class **eventSender**< **savingGroup** >

  Friendship with event sender.

Additional Inherited Members

24.17.1 Detailed Description

Group of saving classes.

Class which defines listens for save triggers from a set of slave classes. Note that this class is itself savable.

24.17.2 Constructor & Destructor Documentation

os::savingGroup::savingGroup ( ) `[inline]`

Default **savingGroup** (p. 526) constructor.

virtual os::savingGroup::~savingGroup ( ) `[inline]`, `[virtual]`

Virtual destructor.

Destructor must be virtual, if an object of this type is deleted, the destructor of the type which inherits this class should be called.

24.17.3 Member Function Documentation

void os::savingGroup::bindSavable (smart_ptr< **savable** > sr)

Checks if a savable class needs to be queued.

Binds a savable class to this receiver checking its current status to see if it needs to be queued for saving.

Parameters

`in`	*sav*	Savable node to be bound

Returns

void

void os::savingGroup::receiveEvent (smart_ptr< **savable** > sr) [protected]

Receives a change event.
This function is triggered by a savable class to which this listener is registered to.

Parameters

in	*sr*	Pointer to savable class

Returns

void

virtual void os::savingGroup::save ( ) [virtual]

Saves the class.
This function must be re-implemented by classes which inherit from the savable class.

Returns

void

Reimplemented from **os::savable** (p. 525).

24.17.4 Friends And Related Function Documentation

friend class eventSender< **savingGroup** > [friend]

Friendship with event sender.
The eventSender must be able to access the **savingGroup::receiveEvent** (p. 528) function.

24.17.5 Member Data Documentation

os::spinLock os::savingGroup::queueLock [private]

Mutex for save queue.

os::smart_ptr<os::unsortedList<**savable**> > os::savingGroup::saveQueue [private]

Queue of savable classes to be re-saved.

24.18 os::Serial Class Reference

This is the **Serial** (p. 528) class. **Serial** (p. 528) objects allow us to abstract out most of the platform irregularities across multiple systems.

Public Member Functions

- **Serial** (char *portName, bool t)

  Serial *(p. 528) constructor Initializes serial communication on a given COM port.*
- virtual ~**Serial** ()

  Serial *(p. 528) destructor Closes a serial connection.*
- int **ReadData** (uint8_t *buffer, uint32_t nbChar)

  Read from a serial port Read data in a buffer, if nbChar is greater than the maximum number of bytes available, it will return only the bytes available. The function return -1 when nothing could be read, the number of bytes actually read.
- bool **WriteData** (uint8_t *buffer, uint32_t nbChar)

  Write to a serial port Writes data from a buffer through the ***Serial*** *(p. 528) connection. Returns true on success.*
- bool **IsConnected** ()

  Check connection Checks on the status of the serial port.

Private Attributes

- uint32_t **hSerial**

  Comm Handler Holds an unsigned integer, which is used for status information.
- bool **connected**

  Connection indicator A boolean which holds whether or not this ***Serial*** *(p. 528) object is connected to a port.*
- bool **track**

  Variable for debugging Allows us to determine whether or not a particular ***Serial*** *(p. 528) object needs to print out debugging information.*

24.18.1 Detailed Description

This is the **Serial** (p. 528) class. **Serial** (p. 528) objects allow us to abstract out most of the platform irregularities across multiple systems.

24.18.2 Constructor & Destructor Documentation

Serial::Serial (char * portName, bool t)

Serial (p. 528) constructor Initializes serial communication on a given COM port.

Serial::~Serial ( ) [virtual]

Serial (p. 528) destructor Closes a serial connection.

24.18.3 Member Function Documentation

bool Serial::IsConnected ( )

Check connection Checks on the status of the serial port.

Returns

bool

int32_t Serial::ReadData (uint8_t * buffer, uint32_t nbChar)

Read from a serial port Read data in a buffer, if nbChar is greater than the maximum number of bytes available, it will return only the bytes available. The function return -1 when nothing could be read, the number of bytes actually read.

Returns

int

bool Serial::WriteData (uint8_t * buffer, uint32_t nbChar)

Write to a serial port Writes data from a buffer through the **Serial** (p. 528) connection. Returns true on success.

Returns

bool

24.18.4 Member Data Documentation

bool os::Serial::connected [private]

Connection indicator A boolean which holds whether or not this **Serial** (p. 528) object is connected to a port.

uint32_t os::Serial::hSerial [private]

Comm Handler Holds an unsigned integer, which is used for status information.

bool os::Serial::track [private]

Variable for debugging Allows us to determine whether or not a particular **Serial** (p. 528) object needs to print out debugging information.

24.19 os::serialThread Class Reference

Serial (p. 528) communication thread The is a serial class that runs as a thread. Thanks to this fact, we can run multiple serial communication threads as well as run a primary set of threads at once.

Public Member Functions

- **serialThread** ()
- **serialThread** (bool track)
- virtual ~**serialThread** ()
- void **serialLoop** (os::smart_ptr< **os::threadHolder** > th)

connection management method This method monitors makes sure that active connections are still active.

- void **sendData** (uint8_t *x, uint32_t nb)

  Sends data Sends data to a given location.
- int32_t **receiveData** (uint8_t *x, uint32_t nb)

  receives data Gets data.

Private Member Functions

- void **build** (bool track)

  Build serial thread This is a method that builds the serial thread and determines if said thread will have debugging statements.
- void **search** (os::smart_ptr< **os::threadHolder** > th)

  Creates serial objects Goes through all possible ports and creates serial objects for those ports which need serial objects.
- void **listen** ()

Private Attributes

- **Serial** * **connection**

  *The connection itself This is a pointer to the actual **Serial*** (p. *528) object.*
- char * **conName**

  Name for the connection Stores the name of this connection.
- std::string * **nameList**

  Stores port names There is a set of possible names for ports. This pointer to string holds the names of various possible ports.
- int32_t **numNames**

  The number of ports This stores the number of unique ports on a given platform.
- int32_t **resetTest**

  Used to detect loss of connection. Stores the number of consecutive failures in sending data. If this variable increases beyond 10, the connection is deleted.
- bool **print**

  Debug variable Used to activate debugging statements.
- bool **active**

  Thread activity Holds if the current serial thread should be active.

24.19.1 Detailed Description

Serial (p. 528) communication thread The is a serial class that runs as a thread. Thanks to this fact, we can run multiple serial communication threads as well as run a primary set of threads at once.

24.19.2 Constructor & Destructor Documentation

serialThread::serialThread ( )

serialThread::serialThread (bool track)

serialThread::~serialThread ( ) [virtual]

24.19.3 Member Function Documentation

void serialThread::build (bool track) [private]

Build serial thread This is a method that builds the serial thread and determines if said thread will have debugging statements.

void os::serialThread::listen ( ) [private]

int32_t serialThread::receiveData (uint8_t * x, uint32_t nb)

receives data Gets data.

Returns

int32_t

void serialThread::search (os::smart_ptr< **os::threadHolder** > th) [private]

Creates serial objects Goes through all possible ports and creates serial objects for those ports which need serial objects.

void serialThread::sendData (uint8_t * x, uint32_t nb)

Sends data Sends data to a given location.

void serialThread::serialLoop (os::smart_ptr< **os::threadHolder** > th)

connection management method This method monitors makes sure that active connections are still active.

24.19.4 Member Data Documentation

bool os::serialThread::active [private]

Thread activity Holds if the current serial thread should be active.

char* os::serialThread::conName [private]

Name for the connection Stores the name of this connection.

Serial* os::serialThread::connection [private]

The connection itself This is a pointer to the actual **Serial** (p. 528) object.

std::string* os::serialThread::nameList [private]

Stores port names There is a set of possible names for ports. This pointer to string holds the names of various possible ports.

int32_t os::serialThread::numNames [private]

The number of ports This stores the number of unique ports on a given platform.

bool os::serialThread::print [private]

Debug variable Used to activate debugging statements.

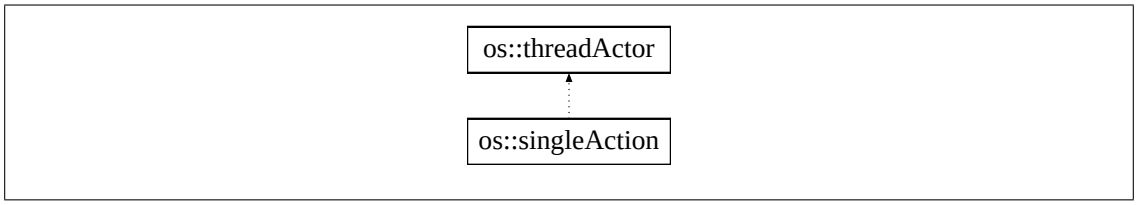
int32_t os::serialThread::resetTest [private]

Used to detect loss of connection. Stores the number of consecutive failures in sending data. If this variable increases beyond 10, the connection is deleted.

24.20 os::singleAction Class Reference

single action class This class is for a thread with only one action.

Inheritance diagram for os::singleAction:



Public Member Functions

- **singleAction** (smart_ptr< **threadDistributor** > dist, void(*rf)(void *), void *rp)
- virtual ~**singleAction** ()
- virtual bool **singleCase** () const

  *indicates singleCase Tells the caller that this **threadActor*** (p. *539) has only one action*
- virtual bool **mustDelete** () const

  *indicates deletion Tells the caller that this **threadActor*** (p. *539) must be deleted.*
- virtual void **perform_action** ()

  calls function Runs the function that was passed when this object was created.
- virtual bool **action_ready** ()

  Indicates readiness Tells the caller that this function is ready to run.

Private Attributes

- void(∗ **recieveFunction**)(void ∗)

  function to run This is a pointer to the function that will run when this thread is called.
- void ∗ **recievePointer**

  function parameters parameters for the thread that will be run.

Additional Inherited Members

24.20.1 Detailed Description

single action class This class is for a thread with only one action.

24.20.2 Constructor & Destructor Documentation

singleAction::singleAction (smart_ptr< **threadDistributor** > dist, void(∗)(void ∗) rf, void ∗ rp)

virtual os::singleAction::~singleAction ( ) [inline], [virtual]

24.20.3 Member Function Documentation

virtual bool os::singleAction::action_ready ( ) [inline], [virtual]

Indicates readiness Tells the caller that this function is ready to run.

Returns

bool

Reimplemented from **os::threadActor** (p. 540).

virtual bool os::singleAction::mustDelete ( ) const [inline], [virtual]

indicates deletion Tells the caller that this **threadActor** (p. 539) must be deleted.

Returns

bool

void singleAction::perform_action ( ) [virtual]

calls function Runs the function that was passed when this object was created.
Reimplemented from **os::threadActor** (p. 540).

virtual bool os::singleAction::singleCase ( ) const [inline], [virtual]

indicates singleCase Tells the caller that this **threadActor** (p. 539) has only one action

Returns

bool

Reimplemented from **os::threadActor** (p. 540).

24.20.4 Member Data Documentation

void(* os::singleAction::recieveFunction) (void *) [private]

function to run This is a pointer to the function that will run when this thread is called.

void* os::singleAction::recievePointer [private]

function parameters parameters for the thread that will be run.

24.21 os::socketTracker Class Reference

socketTracker (p. 535) class Tracks all currently active sockets.

Public Member Functions

- virtual ~**socketTracker** ()
- uint32_t **getNumSockets** ()

  *Gives the number of sockets Returns the number of sockets currently in the **socketTracker** (p. 535).*
- void **closeAll** ()

  Close all sockets Closes all the sockets in the socketHolder.

Static Public Member Functions

- static smart_ptr< **socketTracker** > **singleton** ()

  *Public constructor Creates a **socketTracker** (p. 535) if one does not exists. Returns the **socketTracker** (p. 535) if one does exist.*

Private Member Functions

- void **add** (smart_ptr< **socketUser** > use)

  *Add a socket Adds a socket to the **socketTracker** (p. 535).*
- void **remove** (smart_ptr< **socketUser** > use)

  *Remove a socket Removes a socket from the **socketTracker** (p. 535).*
- **socketTracker** ()

Private Attributes

- **spinLock userLock**

  Lock for safety Ensures safety in multi threaded operation.
- AVLTree< **socketUser** > **users**

  Holds sockets Holds all socket users.

Friends

- class **socketUser**

24.21.1 Detailed Description

socketTracker (p. 535) class Tracks all currently active sockets.

24.21.2 Constructor & Destructor Documentation

socketTracker::socketTracker ( ) [private]

socketTracker::~socketTracker ( ) [virtual]

24.21.3 Member Function Documentation

void socketTracker::add (smart_ptr< **socketUser** > use) [private]

Add a socket Adds a socket to the **socketTracker** (p. 535).

void socketTracker::closeAll ( )

Close all sockets Closes all the sockets in the socketHolder.

uint32_t os::socketTracker::getNumSockets ( ) [inline]

Gives the number of sockets Returns the number of sockets currently in the **socketTracker** (p. 535).

Returns

uint32_t

void socketTracker::remove (smart_ptr< **socketUser** > use) [private]

Remove a socket Removes a socket from the **socketTracker** (p. 535).

smart_ptr< **socketTracker** > socketTracker::singleton ( ) [static]

Public constructor Creates a **socketTracker** (p. 535) if one does not exists. Returns the **socket**↩ **Tracker** (p. 535) if one does exist.

Returns

smart_ptr<socketTracker>

24.21.4 Friends And Related Function Documentation

friend class **socketUser** [friend]

24.21.5 Member Data Documentation

spinLock os::socketTracker::userLock [private]

Lock for safety Ensures safety in multi threaded operation.

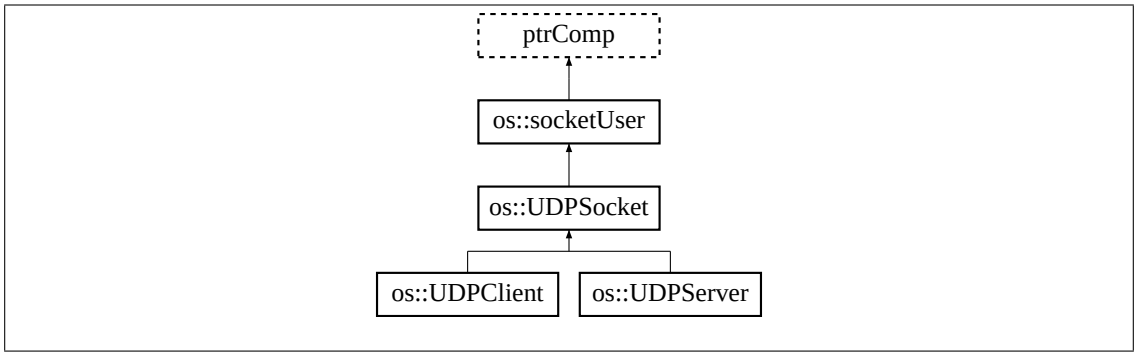
AVLTree<**socketUser**> os::socketTracker::users [private]

Holds sockets Holds all socket users.

24.22 os::socketUser Class Reference

Socket user class This class allows us to manage sockets.

Inheritance diagram for os::socketUser:



Public Member Functions

- **socketUser** ()
- virtual ~**socketUser** ()
- virtual void **openSocket** ()
- virtual void **closeSocket** ()

24.22.1 Detailed Description

Socket user class This class allows us to manage sockets.

24.22.2 Constructor & Destructor Documentation

socketUser::socketUser ( )

socketUser::~socketUser ( ) `[virtual]`

24.22.3 Member Function Documentation

virtual void os::socketUser::closeSocket ( ) `[inline]`, `[virtual]`

Reimplemented in **os::UDPServer** (p. 557), and **os::UDPClient** (p. 551).

virtual void os::socketUser::openSocket ( ) `[inline]`, `[virtual]`

Reimplemented in **os::UDPServer** (p. 558), and **os::UDPClient** (p. 552).

24.23 os::spinLock Class Reference

Public Member Functions

- **spinLock** ()

- virtual ~**spinLock** ()
- void **acquire** ()
- void **release** ()
- bool **isTaken** ()

Private Attributes

- pthread_mutex_t **spinlock**
- bool **taken**

24.23.1 Detailed Description

brief **spinLock** (p. 537) class This is the **spinLock** (p. 537) class we are using. There is nothing particularly complex, but this wraps the available mutex into a cross platform lock. This class has different implementations across different platforms, but the same methods regardless of platform.

24.23.2 Constructor & Destructor Documentation

spinLock::spinLock ( )

spinLock::~spinLock ( ) `[virtual]`

24.23.3 Member Function Documentation

void spinLock::acquire ( )

brief acquires lock This method allows a thread to acquire this **spinLock** (p. 537).

bool spinLock::isTaken ( )

brief indicates lock status This method indicates if the lock is taken, without acquiring the lock.

Returns

bool

void spinLock::release ( )

brief releases lock This method allows a thread to release this **spinLock** (p. 537).

24.23.4 Member Data Documentation

pthread_mutex_t os::spinLock::spinlock `[private]`

brief the base lock This is the base **spinLock** (p. 537). This is the element that changes most significantly across different platforms.

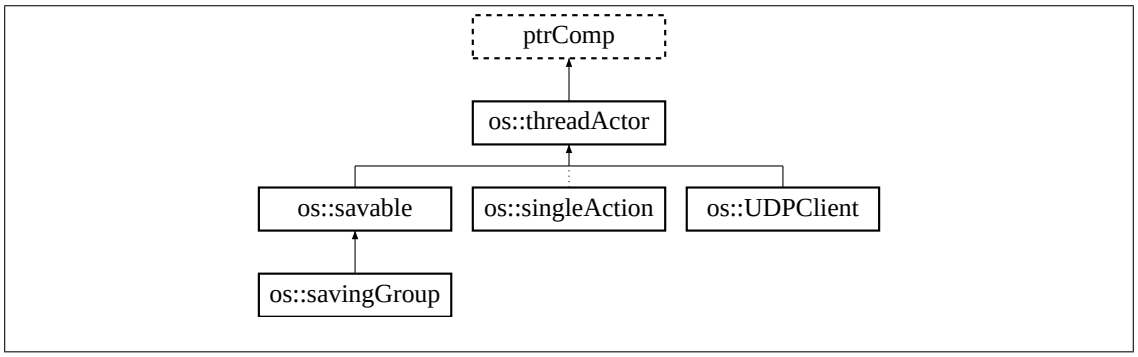
bool os::spinLock::taken `[private]`

brief taken This boolean indicates if the **spinLock** (p. 537) is currently taken.

24.24 os::threadActor Class Reference

threadActor (p. 539) class This class holds information for determining which thread goes at a give time.

Inheritance diagram for os::threadActor:



Public Member Functions

- **threadActor** ()
- virtual ~**threadActor** ()
- void **pushDistributor** (smart_ptr< **threadDistributor** > dist)

  *Adds actor Adds the actor to a given **threadDistributor** (p. 541).*
- void **removeDistributor** ()

  Removes actor from distributor Removes the actor from it's current distributor.
- virtual bool **singleCase** () const

  Indicates if a thread is a single function Indicates if thread is single function.//Access Functions bool isSpecialCase;.
- virtual void **perform_action** ()

  calls thread action If a thread is a single action and has an associated function, that function will be called.
- virtual bool **action_ready** ()

  Indicates if a function is available If the thread is ready to call its function, it will indicate that.

Private Attributes

- smart_ptr< **threadDistributor** > **distributor**

  Distributor for actor This is the distributor for this actor.

Friends

- class **threadDistributor**

24.24.1 Detailed Description

threadActor (p. 539) class This class holds information for determining which thread goes at a give time.

24.24.2 Constructor & Destructor Documentation

threadActor::threadActor ( )

threadActor::~threadActor ( ) [virtual]

24.24.3 Member Function Documentation

virtual bool os::threadActor::action_ready ( ) [inline], [virtual]

Indicates if a function is available If the thread is ready to call its function, it will indicate that.

Returns

bool

Reimplemented in **os::UDPClient** (p. 551), **os::singleAction** (p. 534), and **os::savable** (p. 523).

virtual void os::threadActor::perform_action ( ) [inline], [virtual]

calls thread action If a thread is a single action and has an associated function, that function will be called.

Reimplemented in **os::UDPClient** (p. 552), **os::singleAction** (p. 534), and **os::savable** (p. 525).

void threadActor::pushDistributor (smart_ptr< **threadDistributor** > dist)

Adds actor Adds the actor to a given **threadDistributor** (p. 541).

void threadActor::removeDistributor ( )

Removes actor from distributor Removes the actor from it's current distributor.

virtual bool os::threadActor::singleCase ( ) const [inline], [virtual]

Indicates if a thread is a single function Indicates if thread is single function.//Access Functions bool isSpecialCase;.

uint32_t size;

uint32_t beginning_pos; uint32_t next_pos; std::vector<smart_ptr<logLine> > lineList;

friend class **LineSaverListener** (p. 503); **os::spinLock** (p. 537) lock; smartSet<LineSaverListener> listener;

void priv_addListener(smart_ptr<LineSaverListener> l); void priv_removeListener(smart_ptr<↩ LineSaverListener> l);

@return bool

Reimplemented in **os::singleAction** (p. 534), and **os::savable** (p. 525).

24.24.4 Friends And Related Function Documentation

friend class **threadDistributor** [friend]

24.24.5 Member Data Documentation

smart_ptr<**threadDistributor**> os::threadActor::distributor [private]

Distributor for actor This is the distributor for this actor.

24.25 os::threadDistributor Class Reference

Distributes threads This class allows us to determine which thread should execute at any given time.

Public Member Functions

- **threadDistributor** ()
- **threadDistributor** (uint32_t nt)
- virtual ~**threadDistributor** ()
- void **setNumThreads** (uint32_t nt)

  sets thread count Sets the number of threads allowed in the distributor.
- uint32_t **getNumThreads** () const

  gives thread count Give the number of threads in the exe_thread_list.
- smart_ptr< **threadActor** > **popNext** ()

  Gives next element in the queue Removes the next element in the actor queue that is ready to operate.
- void **pushDone** (smart_ptr< **threadActor** > dn)

  puts an actor onto the queue Places a thread actor onto the actor queue.

Private Member Functions

- void **addActor** (smart_ptr< **threadActor** > act)

  Adds actor Adds an actor to the distributor.
- void **removeActor** (smart_ptr< **threadActor** > act)

  Removes actor Removes an actor from the distributor. It will still be in the queue, but not the list.

Private Attributes

- **spinLock dataLock**

  lock to ensure safe distribution This lock ensures we are operating safely.
- std::queue< smart_ptr< **threadActor** > > **actor_queue**

  holds actors This is a queue of actors. This is for ordering.
- smartSet< **threadActor** > **actor_list**

  holds actors This is a set of actors.
- std::vector< smart_ptr< **executorThread** > > **exe_thread_list**

  holds executors for threads Holds executors for threads, also helps for ordering.

Friends

- class **threadActor**

24.25.1 Detailed Description

Distributes threads This class allows us to determine which thread should execute at any given time.

24.25.2 Constructor & Destructor Documentation

threadDistributor::threadDistributor ( )

threadDistributor::threadDistributor (uint32_t nt)

threadDistributor::~threadDistributor ( ) [virtual]

24.25.3 Member Function Documentation

void threadDistributor::addActor (smart_ptr< **threadActor** > act) [private]

Adds actor Adds an actor to the distributor.

uint32_t os::threadDistributor::getNumThreads ( ) const [inline]

gives thread count Give the number of threads in the exe_thread_list.

Returns

uint32_t

smart_ptr< **threadActor** > threadDistributor::popNext ( )

Gives next element in the queue Removes the next element in the actor queue that is ready to operate.

Returns

smart_ptr<threadActor>

void threadDistributor::pushDone (smart_ptr< **threadActor** > dn)

puts an actor onto the queue Places a thread actor onto the actor queue.

void threadDistributor::removeActor (smart_ptr< **threadActor** > act) [private]

Removes actor Removes an actor from the distributor. It will still be in the queue, but not the list.

void threadDistributor::setNumThreads (uint32_t nt)

sets thread count Sets the number of threads allowed in the distributor.

24.25.4 Friends And Related Function Documentation

friend class **threadActor** [friend]

24.25.5 Member Data Documentation

smartSet<**threadActor**> os::threadDistributor::actor_list [private]

holds actors This is a set of actors.

std::queue<smart_ptr<**threadActor**> > os::threadDistributor::actor_queue [private]

holds actors This is a queue of actors. This is for ordering.

spinLock os::threadDistributor::dataLock [private]

lock to ensure safe distribution This lock ensures we are operating safely.

std::vector<smart_ptr<**executorThread**> > os::threadDistributor::exe_thread_list [private]

holds executors for threads Holds executors for threads, also helps for ordering.

24.26 os::threadHolder Class Reference

Public Member Functions

- **threadHolder** (smart_ptr< std::thread > tp, std::string ti="")
- virtual ~**threadHolder** ()
- void **markFinished** ()

  Indicates that a thread is finished Inside a threaded function, this method is called so that the thread handler knows the status of the thread it is monitoring.
- void **kill** ()

  Notes the thread for killing This method tells the thread handler that the thread is ready to be killed.
- const bool **running** () const

  Gives run status This method simply returns the status of the running boolean.
- const std::string & **threadInfo** () const

  Gives _threadInfo variable Gives the thread information string to the calling object.
- const bool & **killed** ()

  Gives _was_killed variable This gives the kill status of the thread back to the calling object.
- smart_ptr< std::thread > **thread** ()

  Gives a smart pointer to the thread Gives a pointer to the thread for this particular threadHandler to the calling onject.
- const bool **operator==** (const **threadHolder** &th) const
- const bool **operator!=** (const **threadHolder** &th) const
- const bool **operator**<**=** (const **threadHolder** &th) const
- const bool **operator**>**=** (const **threadHolder** &th) const
- const bool **operator**< (const **threadHolder** &th) const
- const bool **operator**> (const **threadHolder** &th) const

Private Attributes

- std::string **_threadInfo**

  Thread data Each thread holder gives a variety of relevant information. The _threadInfo string stores this information.
- smart_ptr< std::thread > **_thread_ptr**

Points at relevant thread Because the thread holder remains in the calling thread, this pointer allows us to know where exactly the thread actually is.

- bool **_running**

  Indicates if a thread is operating This lets a thread check on the operation status of another thread.

- bool **_was_killed**

  Gives status on a thread If a thread must be killed, this boolean indicates the status of the thread.

24.26.1 Constructor & Destructor Documentation

threadHolder::threadHolder (smart_ptr< std::thread > tp, std::string ti = "")

virtual os::threadHolder::~threadHolder ( ) [inline], [virtual]

24.26.2 Member Function Documentation

void os::threadHolder::kill ( ) [inline]

Notes the thread for killing This method tells the thread handler that the thread is ready to be killed.

const bool& os::threadHolder::killed ( ) [inline]

Gives _was_killed variable This gives the kill status of the thread back to the calling object.

Returns

os::threadHolder::_was_killed (p. 545)

void os::threadHolder::markFinished ( ) [inline]

Indicates that a thread is finished Inside a threaded function, this method is called so that the thread handler knows the status of the thread it is monitoring.

const bool os::threadHolder::operator!= (const **threadHolder** & th) const [inline]

const bool os::threadHolder::operator< (const **threadHolder** & th) const [inline]

const bool os::threadHolder::operator<= (const **threadHolder** & th) const [inline]

const bool os::threadHolder::operator== (const **threadHolder** & th) const [inline]

const bool os::threadHolder::operator> (const **threadHolder** & th) const [inline]

const bool os::threadHolder::operator>= (const **threadHolder** & th) const [inline]

const bool os::threadHolder::running ( ) const [inline]

Gives run status This method simply returns the status of the running boolean.

Returns

os::threadHolder::_running (p. 545)

smart_ptr<std::thread> os::threadHolder::thread ( ) [inline]

Gives a smart pointer to the thread Gives a pointer to the thread for this particular threadHandler to the calling onject.

Returns

os::threadHolder::_thread_ptr (p. 545)

const std::string& os::threadHolder::threadInfo ( ) const [inline]

Gives _threadInfo variable Gives the thread information string to the calling object.

Returns

os::threadHolder::_threadInfo (p. 545)

24.26.3 Member Data Documentation

bool os::threadHolder::_running [private]

Indicates if a thread is operating This lets a thread check on the operation status of another thread.

smart_ptr<std::thread> os::threadHolder::_thread_ptr [private]

Points at relevant thread Because the thread holder remains in the calling thread, this pointer allows us to know where exactly the thread actually is.

std::string os::threadHolder::_threadInfo [private]

Thread data Each thread holder gives a variety of relevant information. The _threadInfo string stores this information.

bool os::threadHolder::_was_killed [private]

Gives status on a thread If a thread must be killed, this boolean indicates the status of the thread.

24.27 os::threadTracker Class Reference

Monitors a range of threads This class holds a range of threadHolders. This includes both active and expired threads, ensuring the ability to operate on many threads in mass.

Public Member Functions

- virtual ~**threadTracker** ()
- uint32_t **killTime** () const

  Gives kill time value This method returns the current killTime. killTime is the amount of time a thread can operate before it is automatically killed.
- void **setKillTime** (uint32_t kt)

  *Sets killTime This method allows a user to set the killTime for a given **threadTracker** (p. 545).*

- bool **shutdown** ()

  Kills all threads currently running When a program is ending, this method allows us to kill all threads. If a thread doesn't exit, this will throw an error.
- void **logShutdownFailures** ()

  Logs shut-down failures.
- void **logThreads** ()

  Logging method This method logs thread information into osout.
- void **errorLogThreads** ()

  Logging method This method logs thread information into oserr.
- void **add** (smart_ptr< **threadHolder** > th)

  Add a thread holder Adds a thread holder to the thread tracker. Will error out if the thread cannot be inserted.
- void **remove** (smart_ptr< **threadHolder** > th)

  Remove a thread holder Removes a thread holder from the thread tracker. Will error out if the thread cannot be found.
- uint32_t **getNumThreads** () const

  Gives the number of threads Returns the number of threads in the thread tracker.

Static Public Member Functions

- static smart_ptr< **threadTracker** > **singleton** ()

  *Thread tracker creator Given that we only want one **threadTracker** (p. 545) at any given time, The singleton method checks is a **threadTracker** (p. 545) has already been made. If it has, it retruns a pointer to the **threadTracker** (p. 545). If not, it creates a new **threadTracker** (p. 545) and returns the pointer.*

Private Member Functions

- void **checkKillList** ()

  Removes expired threads When a thread expires, it is not automatically removes. This thread finds and deletes expired threads.
- **threadTracker** ()
- void **log** (smart_ptr< std::ostream > t)

  *Logging method This allows the **threadTracker** (p. 545) to perform logging duties.*

Private Attributes

- AVLTree< **threadHolder** > **killList**

  Tree of threads expired These threads have expired, but not yet deleted.
- AVLTree< **threadHolder** > **threadList**

  Tree of threads running These threads are currently in operation.
- **spinLock lock**

  *Safety **spinLock** (p. 537) This lock ensures that only one thread can update Trees at any given time.*
- uint32_t **_killTime**

  Time before a thread expires Threads can only run so long before we must kill them. This variable allows us to vary how long a thread can run before we kill it.

24.27.1 Detailed Description

Monitors a range of threads This class holds a range of threadHolders. This includes both active and expired threads, ensuring the ability to operate on many threads in mass.

24.27.2 Constructor & Destructor Documentation

threadTracker::threadTracker ( ) [private]

virtual os::threadTracker::~threadTracker ( ) [inline], [virtual]

24.27.3 Member Function Documentation

void threadTracker::add (smart_ptr< **threadHolder** > th)

Add a thread holder Adds a thread holder to the thread tracker. Will error out if the thread cannot be inserted.

void threadTracker::checkKillList ( ) [private]

Removes expired threads When a thread expires, it is not automatically removes. This thread finds and deletes expired threads.

void threadTracker::errorLogThreads ( )

Logging method This method logs thread information into oserr.

uint32_t os::threadTracker::getNumThreads ( ) const [inline]

Gives the number of threads Returns the number of threads in the thread tracker.

Returns

uint32_t

uint32_t os::threadTracker::killTime ( ) const [inline]

Gives kill time value This method returns the current killTime. killTime is the amount of time a thread can operate before it is automatically killed.

Returns

uint32_t

void threadTracker::log (smart_ptr< std::ostream > t) [private]

Logging method This allows the **threadTracker** (p. 545) to perform logging duties.

void threadTracker::logShutdownFailures ( )

Logs shut-down failures.

void threadTracker::logThreads ( )

Logging method This method logs thread information into osout.

void threadTracker::remove (smart_ptr< **threadHolder** > th)

Remove a thread holder Removes a thread holder from the thread tracker. Will error out if the thread cannot be found.

void os::threadTracker::setKillTime (uint32_t kt) [inline]

Sets killTime This method allows a user to set the killTime for a given **threadTracker** (p. 545).

bool threadTracker::shutdown ( )

Kills all threads currently running When a program is ending, this method allows us to kill all threads. If a thread doesn't exit, this will throw an error.

smart_ptr< **threadTracker** > threadTracker::singleton ( ) [static]

Thread tracker creator Given that we only want one **threadTracker** (p. 545) at any given time, The singleton method checks is a **threadTracker** (p. 545) has already been made. If it has, it retruns a pointer to the **threadTracker** (p. 545). If not, it creates a new **threadTracker** (p. 545) and returns the pointer.

Returns

smart_ptr<threadTracker>

24.27.4 Member Data Documentation

uint32_t os::threadTracker::_killTime [private]

Time before a thread expires Threads can only run so long before we must kill them. This variable allows us to vary how long a thread can run before we kill it.

AVLTree<**threadHolder**> os::threadTracker::killList [private]

Tree of threads expired These threads have expired, but not yet deleted.

spinLock os::threadTracker::lock [private]

Safety **spinLock** (p. 537) This lock ensures that only one thread can update Trees at any given time.

AVLTree<**threadHolder**> os::threadTracker::threadList [private]

Tree of threads running These threads are currently in operation.

24.28 os::UDPAVLNode Struct Reference

Public Member Functions

- virtual ~**UDPAVLNode** ()
- const bool **operator==** (const **UDPAVLNode** &comp) const
- const bool **operator**> (const **UDPAVLNode** &comp) const

Public Attributes

- struct sockaddr_in **ipv4_addr**
- struct sockaddr_in6 **ipv6_addr**
- **IPAddress address**

24.28.1 Detailed Description

brief **UDPAVLNode** (p. 549) struct Node used by the UDP server for path rectifying.

24.28.2 Constructor & Destructor Documentation

virtual os::UDPAVLNode::~UDPAVLNode ( ) `[inline]`, `[virtual]`

24.28.3 Member Function Documentation

const bool os::UDPAVLNode::operator== (const **UDPAVLNode** & comp) const `[inline]`

const bool os::UDPAVLNode::operator> (const **UDPAVLNode** & comp) const `[inline]`

24.28.4 Member Data Documentation

IPAddress os::UDPAVLNode::address

brief **IPAddress** (p. 497) of node IP address of the node.

struct sockaddr_in os::UDPAVLNode::ipv4_addr

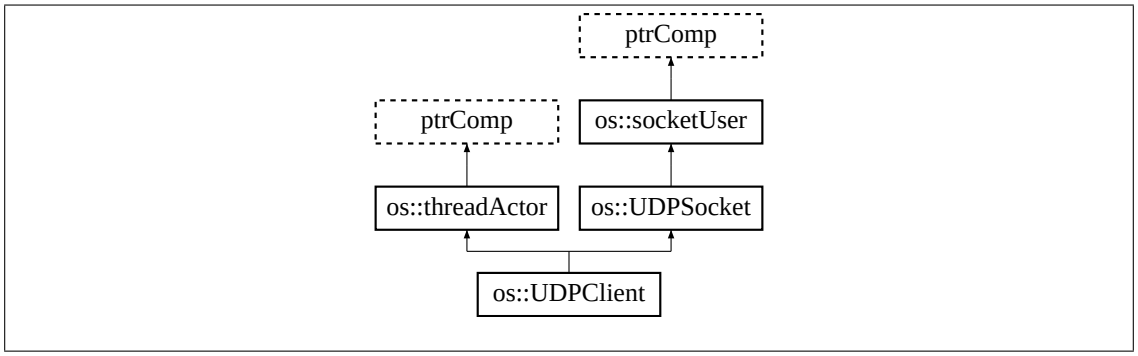
brief IPv4 address IPv4 address of the node.

struct sockaddr_in6 os::UDPAVLNode::ipv6_addr

brief IPv6 address IPv6 address of the node.

24.29 os::UDPClient Class Reference

Inheritance diagram for os::UDPClient:



Public Member Functions

- **UDPClient** (int32_t v4_port, int32_t v6_port, const **IPAddress** &address)
- **UDPClient** (int32_t v4_port, int32_t v6_port, smart_ptr< **myIPAddress** > ip4_requires=NULL)
- virtual ~**UDPClient** ()
- void **openSocket** ()
- void **closeSocket** ()
- void **connect** ()
- void **disconnect** ()
- virtual void **perform_action** ()
- virtual bool **action_ready** ()
- bool **getActive** ()
- bool **getConnected** ()
- void **setReset** (float x)
- bool **send** (smart_ptr< **UDPPacket** > pck)

Private Attributes

- int32_t **intIPv4_Port**
- int32_t **intIPv6_Port**
- bool **active**
- volatile bool **connected**
- float **conTrack**
- float **resetVal**
- **spinLock safeDelete**
- os::smart_ptr< **myIPAddress** > **myIP**
- **IPAddress addr**
- struct sockaddr_in **ipv4_addr**
- struct sockaddr_in6 **ipv6_addr**
- int32_t **s**
- int32_t **slen**
- bool **broadcast**

Additional Inherited Members

24.29.1 Detailed Description

brief UDP Client class This is the class for the UDP Client. The client and server are independent classes.

24.29.2 Constructor & Destructor Documentation

UDPClient::UDPClient (int32_t v4_port, int32_t v6_port, const **IPAddress** & address)

UDPClient::UDPClient (int32_t v4_port, int32_t v6_port, smart_ptr< **myIPAddress** > ip4_requires = `NULL`)

UDPClient::~UDPClient ( ) `[virtual]`

24.29.3 Member Function Documentation

bool UDPClient::action_ready ( ) `[virtual]`

brief checks if a message is available. Checks if a message is available.

Returns

bool

Reimplemented from **os::threadActor** (p. 540).

void UDPClient::closeSocket ( ) `[virtual]`

brief closes socket Closes the socket of the client.
Reimplemented from **os::socketUser** (p. 537).

void UDPClient::connect ( )

brief begins connection Initiates the connection for this client.

void UDPClient::disconnect ( )

brief forces disconnection Forcibly closes the connection.

bool UDPClient::getActive ( ) `[virtual]`

brief gives activity status. Gives the active boolean to caller.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

bool UDPClient::getConnected ( ) [virtual]

brief gives connection status Calculates if the client is currently connected.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

void UDPClient::openSocket ( ) [virtual]

brief opens socket Opens the socket of the client.
Reimplemented from **os::socketUser** (p. 537).

void UDPClient::perform_action ( ) [virtual]

brief receives a message Receives a message.
Reimplemented from **os::threadActor** (p. 540).

bool UDPClient::send (smart_ptr< **UDPPacket** > pck) [virtual]

brief sends data Attempts to send data. Will report on the success or failure of the transmission.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

void UDPClient::setReset (float x)

brief sets reset Sets the amount of time time allowed before the connection is considered dead.

24.29.4 Member Data Documentation

bool os::UDPClient::active [private]

brief indicates if active Indicates if this client is active.

IPAddress os::UDPClient::addr [private]

brief address of target Holds the IP address of the target device.

bool os::UDPClient::broadcast [private]

brief indicates broadcast mode Stores if the client is broadcasting.

volatile bool os::UDPClient::connected [private]

brief indicates if connected Indicates if this client is currently connected.

float os::UDPClient::conTrack [private]

brief holds time Holds the time for determining connection status.

int32_t os::UDPClient::intIPv4_Port [private]

brief port for IPv4 Holds the port for IPv4 transmissions.

int32_t os::UDPClient::intIPv6_Port [private]

brief port for IPv6 Holds the port for IPv6 transmissions.

struct sockaddr_in os::UDPClient::ipv4_addr [private]

brief IPv4 address of target Holds the IPv4 address of the target device.

struct sockaddr_in6 os::UDPClient::ipv6_addr [private]

brief IPv6 address of target Holds the IPv6 address of the target device.

os::smart_ptr<**myIPAddress**> os::UDPClient::myIP [private]

brief client's IP address Holds the client's own IP address

float os::UDPClient::resetVal [private]

brief holds timeout Holds the amount of time that must pass before a connection is considered inactive.

int32_t os::UDPClient::s [private]

brief socket Socket used in data transmission.

spinLock os::UDPClient::safeDelete [private]

brief lock for safety Ensures that the client can be safely multi threaded.

int32_t os::UDPClient::slen [private]

brief length of address of target socket This is the length of the address of the target socket.

24.30 os::UDPPacket Class Reference

Public Member Functions

- **UDPPacket** (uint8_t *input, const **IPAddress** &i, int32_t p)
- **UDPPacket** (uint8_t *output, int32_t l, int32_t t, const **IPAddress** &i, int32_t p)
- virtual ~**UDPPacket** ()
- int32_t **getLength** () const

- int32_t **getType** () const
- uint8_t * **getData** ()
- const **IPAddress** & **getAddress** () const
- int32_t **getPort** () const
- os::smart_ptr< byte > **sendData** () const

Private Attributes

- **IPAddress ip**
- bool **in_or_out**
- int32_t **port**
- uint8_t * **data**
- uint16_t **length**
- uint8_t **type**

24.30.1 Detailed Description

brief **UDPPacket** (p. 553) class This is the UDP Packet class, it can be sent and received.

24.30.2 Constructor & Destructor Documentation

UDPPacket::UDPPacket (uint8_t * input, const **IPAddress** & i, int32_t p)

brief receiving initializer Sets up a packet to be received.

UDPPacket::UDPPacket (uint8_t * output, int32_t l, int32_t t, const **IPAddress** & i, int32_t p)

brief sending initializer Sets up a packet to be sent.

UDPPacket::~UDPPacket ( ) [virtual]

24.30.3 Member Function Documentation

const **IPAddress** & UDPPacket::getAddress ( ) const

brief gets IP address Returns the IP address of the packet.

Returns

smart_ptr<IPAddress>

uint8_t * UDPPacket::getData ( )

brief gets data Returns the data from the packet.

Returns

uint8_t*

int32_t UDPPacket::getLength ( ) const

brief gets length Returns the length of the packet.

Returns

int32_t

int32_t UDPPacket::getPort ( ) const

brief gets port Returns the port of the packet.

Returns

int32_t

int32_t UDPPacket::getType ( ) const

brief gets type Returns the type of the packet.

Returns

int32_t

os::smart_ptr< byte > UDPPacket::sendData ( ) const

brief preps data for transmission Prepares the data for transmission. This packs the type, length, and data into a single variable.

Returns

uint8_t*

24.30.4 Member Data Documentation

uint8_t* os::UDPPacket::data [private]

brief packet contents Holds the information this packet is transmitting.

bool os::UDPPacket::in_or_out [private]

brief indicates direction of data flow This value is true if the packet is being received. This value is false if this packet is being received.

IPAddress os::UDPPacket::ip [private]

brief IP address of target The location of the target of this packet.

uint16_t os::UDPPacket::length [private]

brief length of transmitted data This holds the length of the data in the packet.

int32_t os::UDPPacket::port [private]

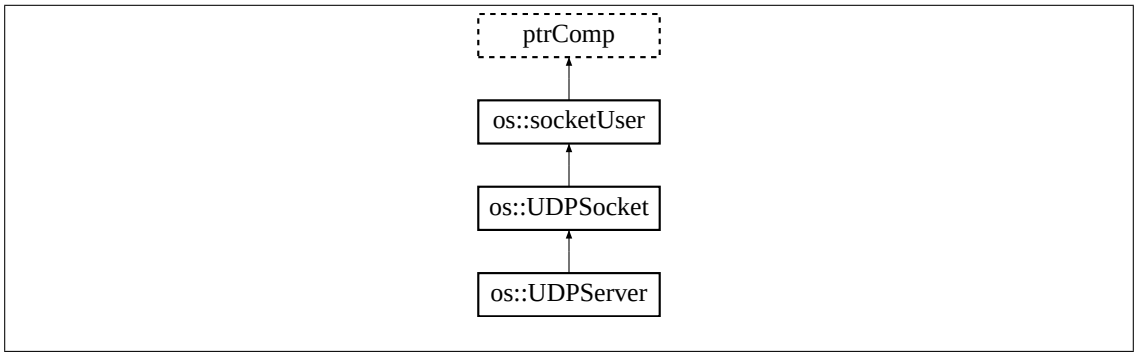
brief port for connection Holds the port for the packet.

uint8_t os::UDPPacket::type [private]

brief type of packet Stores the type of the packet.

24.31 os::UDPServer Class Reference

Inheritance diagram for os::UDPServer:



Public Member Functions

- **UDPServer** (int32_t v4_port, int32_t v6_port)
- virtual ~**UDPServer** ()
- void **openSocket** ()
- void **closeSocket** ()
- void **start** ()
- void **end** ()
- void **receiveLoopIPV4** (smart_ptr< **threadHolder** > th)
- void **receiveLoopIPV6** (smart_ptr< **threadHolder** > th)
- bool **getActive** ()
- bool **send** (smart_ptr< **UDPPacket** > pck)
- bool **getConnected** ()

Private Attributes

- int32_t **intIPv4_Port**
- int32_t **intIPv6_Port**
- bool **ip4_active**
- bool **ip6_active**
- volatile bool **connected**
- **spinLock safeDelete**
- AVLTree< **UDPAVLNode** > **ipAddressRef**

- **spinLock avlLock**
- struct sockaddr_in **ipv4_addr**
- struct sockaddr_in6 **ipv6_addr**
- int32_t **ip4_soc**
- int32_t **ip6_soc**
- socklen_t **ip4_len**
- socklen_t **ip6_len**

Additional Inherited Members

24.31.1 Detailed Description

brief **UDPServer** (p. 556) class This is the class for the server half of our socket connection.

24.31.2 Constructor & Destructor Documentation

UDPServer::UDPServer (int32_t v4_port, int32_t v6_port)

UDPServer::~UDPServer ( ) [virtual]

24.31.3 Member Function Documentation

void UDPServer::closeSocket ( ) [virtual]

brief closes sockets Closes the server's sockets.

Reimplemented from **os::socketUser** (p. 537).

void UDPServer::end ( )

brief shuts down the server Ends current connections and closes the active sockets.

bool UDPServer::getActive ( ) [virtual]

brief gets active Returns the active status to the caller.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

bool UDPServer::getConnected ( ) [virtual]

brief gets the connection status Gives the connection status to the caller.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

void UDPServer::openSocket ( ) [virtual]

brief opens sockets Opens the server's sockets.
Reimplemented from **os::socketUser** (p. 537).

void UDPServer::receiveLoopIPV4 (smart_ptr< **threadHolder** > th)

brief receive loop for IPv4 This method is intended to run in it's own thread, where it will read in information. This method adds clients to the AVL tree of nodes. It also calls the received method method so that other threads know that new data has arrived.

void UDPServer::receiveLoopIPV6 (smart_ptr< **threadHolder** > th)

brief receive loop for IPv6 This method is intended to run in it's own thread, where it will read in information. This method adds clients to the AVL tree of nodes. It also calls the received method method so that other threads know that new data has arrived.

bool UDPServer::send (smart_ptr< **UDPPacket** > pck) [virtual]

brief sends a packet This method sends a provided **UDPPacket** (p. 553) to its target.

Returns

bool

Reimplemented from **os::UDPSocket** (p. 561).

void UDPServer::start ( )

brief starts the server This method starts the server by creating the necessary sockets and starting the listening threads for IPv4 and IPv6 transmission.

24.31.4 Member Data Documentation

spinLock os::UDPServer::avlLock [private]

brief tree lock Allows us to access the tree of nodes while multi-threading.

volatile bool os::UDPServer::connected [private]

brief server connection status Indicates if the server is currently connected to a client.

int32_t os::UDPServer::intIPv4_Port [private]

brief IPv4 address of server This is the IPv4 address of the server. (This node).

int32_t os::UDPServer::intIPv6_Port [private]

brief IPv6 address of the server This is the IPv6 address of the server. (This node).

bool os::UDPServer::ip4_active [private]

brief IPv4 activity Indicates if the IPv4 address is currently active.

socklen_t os::UDPServer::ip4_len [private]

brief size of target socket This is the size of the target's IPv4 socket.

int32_t os::UDPServer::ip4_soc [private]

brief IPv4 socket Address of the IPv4 socket for this server.

bool os::UDPServer::ip6_active [private]

brief IPv6 activity Indicates if the IPv6 address is currently active.

socklen_t os::UDPServer::ip6_len [private]

brief size of target socket This is the size of the target's IPv6 socket.

int32_t os::UDPServer::ip6_soc [private]

brief IPv6 socket Address of the IPv6 socket for this server.

AVLTree<**UDPAVLNode**> os::UDPServer::ipAddressRef [private]

brief tree of nodes This is a tree of target nodes that this server knows of.

struct sockaddr_in os::UDPServer::ipv4_addr [private]

brief IPv4 address of target This is the IPv4 address of the current client of this server.

struct sockaddr_in6 os::UDPServer::ipv6_addr [private]

brief IPv6 address of target This is the IPv6 address of the current client of this server.

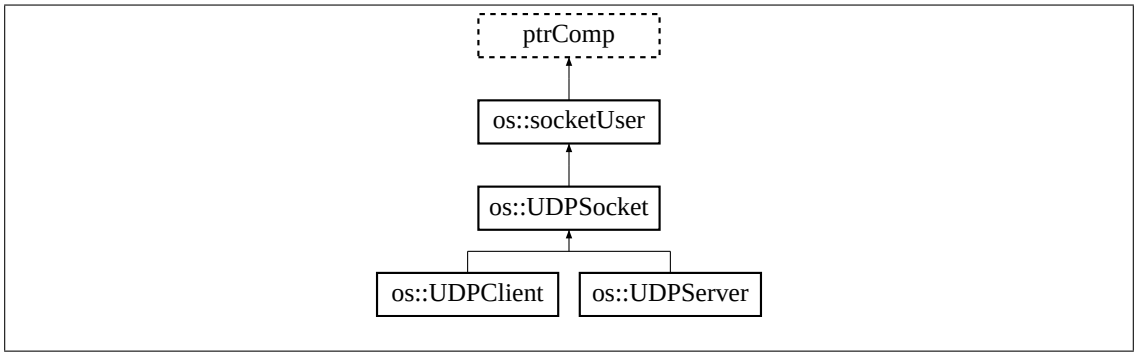
spinLock os::UDPServer::safeDelete [private]

brief deletion lock This lock allows us to safely delete things while multi-threading.

24.32 os::UDPSocket Class Reference

UDPSocket (p. 559) class A class for UDPSockets, which in turn allows us to multi thread the packet send/receive functionality.

Inheritance diagram for os::UDPSocket:



Public Member Functions

- **UDPSocket** ()
- virtual ~**UDPSocket** ()
- virtual bool **getActive** ()

  returns state of socket. Gives the status of the current socket.
- void **setReceiveEvent** (void(∗func)(void ∗), void ∗ptr)

  Sets receive functionality Determines what function and parameters will be activated when the receive event occurs.
- virtual bool **send** (smart_ptr< **UDPPacket** > pck)

  *Send a **UDPPacket** (p. 553) Sends a **UDPPacket** (p. 553) and returns whether or not the packet was sent successfully.*
- virtual bool **getConnected** ()

  Gives connection status Returns the status of the socket.
- bool **available** ()

  Indicates if a packet can be received Lets the caller know if there are packets available to read.
- smart_ptr< **UDPPacket** > **receive** ()

  Gives the next packet Gives the next packet in the queue.

Protected Attributes

- **safeQueue**< **UDPPacket** > **incomingPackets**

  Incoming data A queue of incoming packets. This is thread safe so we can run things concurrently.
- **spinLock popLock**

  Lock for popping. Ensures we can safely pop the packet queue.
- void(∗ **receiveFunction**)(void ∗)

  Called on packet received Determines what function will be called when a packet is received.
- void ∗ **receivePointer**

  Parameters for receiveFunction Parameters for the called function.

24.32.1 Detailed Description

UDPSocket (p. 559) class A class for UDPSockets, which in turn allows us to multi thread the packet send/receive functionality.

24.32.2 Constructor & Destructor Documentation

UDPSocket::UDPSocket ( )

UDPSocket::~UDPSocket ( ) [virtual]

24.32.3 Member Function Documentation

bool os::UDPSocket::available ( ) [inline]

Indicates if a packet can be received Lets the caller know if there are packets available to read.

Returns

bool

virtual bool os::UDPSocket::getActive ( ) [inline], [virtual]

returns state of socket. Gives the status of the current socket.

Returns

bool

Reimplemented in **os::UDPServer** (p. 557), and **os::UDPClient** (p. 551).

virtual bool os::UDPSocket::getConnected ( ) [inline], [virtual]

Gives connection status Returns the status of the socket.

Returns

bool

Reimplemented in **os::UDPServer** (p. 557), and **os::UDPClient** (p. 552).

smart_ptr< **UDPPacket** > UDPSocket::receive ( )

Gives the next packet Gives the next packet in the queue.

Returns

smart_ptr<UDPPacket>

virtual bool os::UDPSocket::send (smart_ptr< **UDPPacket** > pck) [inline], [virtual]

Send a **UDPPacket** (p. 553) Sends a **UDPPacket** (p. 553) and returns whether or not the packet was sent successfully.

Returns

bool

Reimplemented in **os::UDPServer** (p. 558), and **os::UDPClient** (p. 552).

void UDPSocket::setReceiveEvent (void(*)(void *) func, void * ptr)

Sets receive functionality Determines what function and parameters will be activated when the receive event occurs.

24.32.4 Member Data Documentation

safeQueue<**UDPPacket**> os::UDPSocket::incomingPackets [protected]

Incoming data A queue of incoming packets. This is thread safe so we can run things concurrently.

spinLock os::UDPSocket::popLock [protected]

Lock for popping. Ensures we can safely pop the packet queue.

void(* os::UDPSocket::receiveFunction) (void *) [protected]

Called on packet received Determines what function will be called when a packet is received.

void* os::UDPSocket::receivePointer [protected]

Parameters for receiveFunction Parameters for the called function.

24.33 os::USBFile Class Reference

Public Member Functions

- **USBFile** ()
- virtual ~**USBFile** ()
- bool **isUSBDrive** ()
- **USBNode** * **getCurrentDrive** ()

24.33.1 Constructor & Destructor Documentation

USBFile::USBFile ( )

USBFile::~USBFile ( ) [virtual]

24.33.2 Member Function Documentation

USBNode * USBFile::getCurrentDrive ( )

bool USBFile::isUSBDrive ( )

24.34 os::USBNode Class Reference

This class stores the location of a USB device.

Public Member Functions

- **USBNode** (std::string p)
- std::string **getPath** ()

  Getter for the path string. This is a simple getter, it returns a string.

Private Attributes

- std::string **path**

  USB path This string holds the path to a USB device.

24.34.1 Detailed Description

This class stores the location of a USB device.

24.34.2 Constructor & Destructor Documentation

USBNode::USBNode (std::string p)

24.34.3 Member Function Documentation

std::string USBNode::getPath ( )

Getter for the path string. This is a simple getter, it returns a string.

Returns

std::string::path

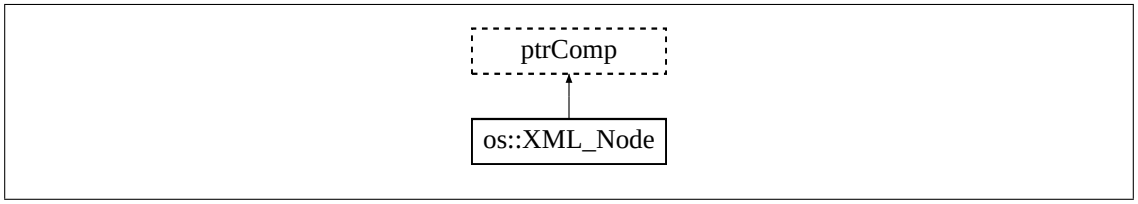
24.34.4 Member Data Documentation

std::string os::USBNode::path `[private]`

USB path This string holds the path to a USB device.

24.35 os::XML_Node Class Reference

XML Node class The core node of our XML parsing.
Inheritance diagram for os::XML_Node:



Public Member Functions

- **XML_Node** (std::string _id)
- virtual ~**XML_Node** ()
- void **setData** (std::string str)

  sets data Sets the data element to a given string.
- **smartXMLNodeList findElement** (std::string _id)

  finds node Returns the a list with each node that has the given identifier.
- void **deleteElement** (std::string _id)

  removes element Removes all nodes with a given identifier and their children.
- void **addElement** (**smartXMLNode** node)

  adds element Adds an element to the children of the current node.
- const std::string & **getID** () const

  gets identifier Returns the identifier of a node.
- const std::string & **getData** () const

  gets data Returns the data of a node.
- **smartXMLNodeList** & **getChildren** ()

  returns children Returns the children of a node.
- const **smartXMLNodeList getChildren** () const

  returns children Returns the children of a node.
- const std::vector< std::string > & **getDataList** () const

  returns dataList Returns the dataLust of a node.
- std::vector< std::string > & **getDataList** ()

  returns dataList Returns the dataLust of a node.

Private Attributes

- std::string **ID**

  node identifier Identifies the node with a specific name.
- std::string **data**

  Holds datum Holds one piece of data, assuming this node holds only one piece of data.
- **smartXMLNodeList children**

  children of node The children of this node, as this system is structured as a tree.
- std::vector< std::string > **dataList**

  Holds data Holds a list of data, assuming we have more than one piece of data to store in this node.

24.35.1 Detailed Description

XML Node class The core node of our XML parsing.

24.35.2 Constructor & Destructor Documentation

XML_Node::XML_Node (std::string _id)

virtual os::XML_Node::~XML_Node ( ) [inline], [virtual]

24.35.3 Member Function Documentation

void XML_Node::addElement (**smartXMLNode** node)

adds element Adds an element to the children of the current node.

void XML_Node::deleteElement (std::string _id)

removes element Removes all nodes with a given identifier and their children.

smartXMLNodeList XML_Node::findElement (std::string _id)

finds node Returns the a list with each node that has the given identifier.

Returns

smart_ptr<unsortedList<XMLNode> >

smartXMLNodeList& os::XML_Node::getChildren ( ) [inline]

returns children Returns the children of a node.

Returns

smart_ptr<unsortedList<XMLNode> >&

const **smartXMLNodeList** os::XML_Node::getChildren ( ) const [inline]

returns children Returns the children of a node.

Returns

const smart_ptr<unsortedList<XMLNode> >

const std::string& os::XML_Node::getData ( ) const [inline]

gets data Returns the data of a node.

Returns

const std::string&

const std::vector<std::string>& os::XML_Node::getDataList ( ) const [inline]

returns dataList Returns the dataLust of a node.

Returns

const std::vector<std::string>&

std::vector<std::string>& os::XML_Node::getDataList ( ) [inline]

returns dataList Returns the dataLust of a node.

Returns

std::vector<std::string>&

const std::string& os::XML_Node::getID ( ) const [inline]

gets identifier Returns the identifier of a node.

Returns

const std::string&

void os::XML_Node::setData (std::string str) [inline]

sets data Sets the data element to a given string.

24.35.4 Member Data Documentation

smartXMLNodeList os::XML_Node::children [private]

children of node The children of this node, as this system is structured as a tree.

std::string os::XML_Node::data [private]

Holds datum Holds one piece of data, assuming this node holds only one piece of data.

std::vector<std::string> os::XML_Node::dataList [private]

Holds data Holds a list of data, assuming we have more than one piece of data to store in this node.

std::string os::XML_Node::ID [private]

node identifier Identifies the node with a specific name.