



M5T Framework SAFE v2.1 - API Reference

Proprietary & Confidential

Legal Information

Copyright © 2010 Media5 Corporation ("Media5")

This document contains information that is confidential and proprietary to Media5.

Media5 reserves all rights to this document as well as to the Intellectual Property of the document and the technology and know-how that it includes and represents.

This publication cannot be reproduced, neither in whole nor in part, in any form whatsoever, without prior written approval by Media5.

Media5 reserves the right to revise this publication and make changes at any time and without the obligation to notify any person and/or entity of such revisions and/or changes.

Document Build Date

This document was built on: May 26, 2010

Table of Contents

Introduction	1
M5T Framework	2
General Tracing Configuration	2
Assertion Mechanism Overview	5
M5T Framework tracing nodes	6
g_stTraceRoot Variable	7
g_stFramework Variable	7
Basic	10
Enumerations	10
Functions	17
Macros	37
Deprecated Macros	62
Structures	62
Templates	64
Types	84
Variables	94
Cap	94
Classes	94
Structures	157
Templates	158
Config	261
Macros	261
Variables	318
Crypto	318
Classes	318
Enumerations	409
Variables	411
ECom	412
Classes	413
Functions	420
Macros	423
Types	425
Kerberos	426
Classes	426
Enumerations	463

Kernel	464
Classes	464
Enumerations	504
Functions	506
Macros	509
Structures	511
Templates	515
Types	521
Variables	521
Network	522
Classes	522
Enumerations	650
Functions	652
Structures	662
Variables	663
Pki	663
Classes	663
RegExp (Regular Expressions)	738
Classes	738
Structures	740
Resolver	741
Classes	741
Structures	762
ServicingThread	762
Classes	762
Enumerations	789
Startup	789
Classes	789
Structures	791
Variables	794
Tls	794
Classes	794
Enumerations	815
Time	816
Classes	816
Enumerations	838
Variables	839
Xml	839
Classes	839

Enumerations	883
--------------	-----

Index	884
--------------	------------

1 - Introduction

Welcome to the M5T Framework SAFE v2.1 API Reference. This document provides quick and simple references to the usage of the M5T Framework. It should be used by programmers who already have some knowledge about the M5T Framework.

All modifications to this API Reference are tracked through CRs (Change Requests) and are documented in the release notes associated with this product.

The M5T Framework SAFE v2.1 still supports a number of APIs that are deprecated. A deprecated API means that it is accessible in this version of the software, but will be eventually removed, so its use is discouraged.

The following are the deprecated APIs of the M5T Framework SAFE v2.1:

- `MxTraceDisableLevel` ([see page 24](#))()
- `MxTraceEnableLevel` ([see page 26](#))()
- `MxTraceEnableMaxLevels` ([see page 26](#))()
- `MxTraceGetLevelsEnabled` ([see page 29](#))()
- Deprecated Macros ([see page 62](#))
- Deprecated Data Types section ([see page 85](#))
- `ISocket::GetProtocolFamily` ([see page 648](#))

2 - M5T Framework

This section describes the content of the M5T Framework. The documentation is divided into subsections, each of which describes one specific part of the M5T Framework. The subsection's names are identical to the source folder they document. There are special sections that describe more specific features of the M5T Framework. These sections are listed before any folder specific documentation.

- General Tracing configuration (see page 2)
- Assertion Mechanism Overview (see page 5)
- Framework tracing nodes (see page 6)
- Basic (see page 10)
- Cap (see page 94)
- Config (see page 261)
- Crypto (see page 318)
- ECom (see page 412)
- Kerberos (see page 426)
- Kernel (see page 464)
- Network (see page 522)
- Pki (see page 663)
- RegExp (Regular Expressions) (see page 738)
- Resolver (see page 741)
- ServicingThread (see page 762)
- Startup (see page 789)
- Tls (see page 794)
- Time (see page 816)
- Xml (see page 839)

2.1 - General Tracing Configuration

The Framework offers an extensive tracing mechanism with its various software packages. The goals of the tracing mechanism are to:

- Provide multiple tracing levels. Each level can be enabled or disabled at compile time. When a level is disabled at compile time, the associated tracing macro expands to nothing. When a level is enabled at compile time, the level can be enabled or disabled at run time.
- Provide the possibility to create any number of types or subtypes of traces using tracing nodes. Each trace node can be enabled or disabled at run time.
- Augment the traces with meta-information (task id, time stamp, etc.).
- Use a format to ease automatic parsing.
- Be flexible and extensible.

The tracing mechanism is made up of five main parts:

- Trace level part that can be configured at compile time and run time.
- Trace node part that can only be configured at run time.
- Trace output handlers that can be configured at compile time and run time.
- Trace format handlers that can be configured at compile time and run time.
- Trace call stack handlers that can be configured at compile time and run time.

Before describing the details of each of the above building blocks, it is important to highlight how the tracing mechanism determines if a trace has to be outputted. Two conditions **MUST** be present to have a trace displayed. The level of the tracing macro used and the node used as a parameter of the tracing macro **MUST** be enabled. There is no direct relationship between trace's levels and the trace nodes in the sense that any level can be applied to any trace node.

The following paragraphs present a detailed description of the tracing mechanism main parts.

Tracing Levels

The framework defines ten tracing levels ordered from eLEVEL0 (Highest severity) to eLEVEL9 (lowest severity). There are two complementary macros that regroup all or none of the levels. See the complete list of levels in the EMxTraceLevel (see page 16) documentation.

Tracing levels can be configured at compile time using various pre-processor defines. Tracing levels are also configurable at run time using different methods. The compile time and run time configuration capabilities of the tracing levels help to better suit any application needs.

Compile time configuration is possible to determine which trace levels are compiled-in using a list of pre-processor defines. Each level is controlled independently but it is also possible to control multiple levels at once. The pre-processor define of a given level activates three tracing macros. For example, MXD_TRACE0_ENABLE_SUPPORT (see page 314) enables MX_TRACE0 (see page 52), MX_TRACE0_HEX (see page 54), and MX_TRACE0_IS_ENABLED (see page 54). The complete list of pre-processor defines is as follows:

- MXD_TRACE0_ENABLE_SUPPORT (see page 314)
- MXD_TRACE1_ENABLE_SUPPORT (see page 314)
- MXD_TRACE2_ENABLE_SUPPORT (see page 314)
- MXD_TRACE3_ENABLE_SUPPORT (see page 314)
- MXD_TRACE4_ENABLE_SUPPORT (see page 314)
- MXD_TRACE5_ENABLE_SUPPORT (see page 314)
- MXD_TRACE6_ENABLE_SUPPORT (see page 314)
- MXD_TRACE7_ENABLE_SUPPORT (see page 314)
- MXD_TRACE8_ENABLE_SUPPORT (see page 314)
- MXD_TRACE9_ENABLE_SUPPORT (see page 314)
- MXD_TRACE_ENABLE_ALL (see page 314)
- MXD_TRACE_MAX_LEVEL (see page 314)

Run time configuration of the tracing levels is possible regardless of the compile time configuration used. At run time, each level can be controlled independently on a per node basis or for all the nodes in a subtree. It is also possible to select multiple levels at the same time. The following is the list of methods available for run time configuration of the tracing levels:

- MxTraceEnableNodeLevel (see page 27)
- MxTraceDisableNodeLevel (see page 25)
- MxTraceEnableNodeMaxLevels (see page 28)
- MxTraceGetNodeLevelsEnabled (see page 29)

Tracing Nodes

A tracing node (STrakeNode (see page 64)) is located in a tree and represents a logical entity to regroup traces. A node can have zero or more child and it has its own state that is independent from the state of other nodes. Like the trace levels, a node can be configured to be enabled or disabled. Any number of nodes can exist in the system.

Since there is no limit on the number of tracing nodes, access to a node is designed differently for the configuration of its state and for issuing a trace. When issuing a trace, the tracing node variable is used directly. Programmatically, the visibility of a trace node variable is meant to be restricted to the part of code using it. On the other hand, enabling or disabling a tracing node might come from a totally different module in the program where the tracing node variable is not available. Instead, referring a node is possible using a string in the form of a path like in directory trees. This explains why the nodes MUST be grouped in a tree structure.

The tree is built at run time with registration and un-registration methods. At the beginning, only the tracing mechanism root node exists. The root node is the only one that needs to be published so children can be attached to it. The root node is accessible through its node variable, which is called g_stTraceRoot (see page 7). Children are added during the initialization process to create the tree. Usually, children of the root node represent high-level logical grouping. For example, the Framework node and SIP stack node are children of the root node. In turn, these high-level grouping can contain children representing more precise grouping. For example, the Framework has many folders: Network, Kernel, etc. The Kernel folder in turn has many classes CMutex (see page 488), CThread (see page

491), etc. The methods for registering and un-registering nodes are listed below:

- [MxTraceRegisterNode](#) (see page 30)
- [MxTraceUnregisterNode](#) (see page 34)

The nodes allowed to trace are configured at run time using two methods. Each node can be controlled independently. The tree structure gives additional flexibility to the run time configuration methods through a flag indicating that recursion is required. The string used to refer to the root node is "/". The methods for enabling and disabling nodes are listed below:

- [MxTraceEnableNode](#) (see page 27)
- [MxTraceDisableNode](#) (see page 24)

Trace Output Handlers

The following output handlers are available to send the traces to different mediums.

- [MxTraceToStderr](#) (see page 33)
- [MxTraceToStdout](#) (see page 34)
- [MxTraceToDebugger](#) (see page 33)

An application can also define its own output handler by defining a function that follows the [mxt_PFNTraceOutputHandler](#) (see page 92) typedef and calling [MxTraceAddOutputHandler](#) (see page 23) or [MxTraceSetNewOutputHandler](#) (see page 32) or by overriding the default handler with [MXD_TRACE_OUTPUT_HANDLER_OVERRIDE](#) (see page 311).

Trace Format Handlers

The format handler is the function that processes the trace and puts it in a pre-defined format before the result is given to the actual output handler. The Framework provides a default format handler. Refer to [mxt_PFNTraceFormatHandler](#) (see page 89) for a complete description of the default format handler. The following methods provide run time control over the default format handler configuration:

- [MxTraceEnableDefaultFields](#) (see page 26): Enables default tracing fields.
- [MxTraceEnableSyslogFields](#) (see page 28): Enables syslog tracing fields.
- [MxTraceEnableAllFields](#) (see page 25): Enables all tracing fields.
- [MxTraceDisableAllFields](#) (see page 24): Disables all tracing fields.
- [MxTraceEnableField](#) (see page 26): Enables a specific tracing field.
- [MxTraceDisableField](#) (see page 24): Disables a specific tracing field.

The default format handler can be overridden with an application format handler by defining [MXD_TRACE_FORMAT_HANDLER_OVERRIDE](#) (see page 310) to a function that follows the [mxt_PFNTraceFormatHandler](#) (see page 89) type.

```
// This code should be included in the PreMxConfig.h configuration file.
//-----
#define MXD_TRACE_FORMAT_HANDLER_OVERRIDE MyFormatHandler.
```

It can also be overridden at run time using the [MxTraceSetNewFormatHandler](#) (see page 32) method.

Call Stack Handlers

For a limited OS/Architecture combinations, it is possible to print the function call stack traces. Call stack handlers are used for this purpose. They can be configured at compile time using the [MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE](#) (see page 309) and [MXD_TRACE_ENABLE_MACRO_CALLSTACK](#) (see page 314) or at run time using [MxTraceSetNewCallStackHandler](#) (see page 31).

Finally, the framework offers other compile time and run time configuration macros and methods.

Other Compile Time Configuration Macros

- [MXD_TRACE_BUFFER_CAPACITY](#) (see page 309): Defines the maximal trace size.
- [MXD_TRACE_PROGNAME](#) (see page 312): Defines a program name to be prefixed to traces.

- MXD_TRACE_BACKTRACE_CAPACITY (see page 309): Defines the backtrace buffer for Linux.
- In addition, a new trace buffering mechanism may be defined with one of the following macros:
MXD_TRACE_USE_STACK_BUFFER (see page 313), MXD_TRACE_USE_DYNAMIC_ALLOC_BUFFER (see page 313),
MXD_TRACE_USE_MEMORY_QUEUE_BUFFER (see page 313).

M5T product's internal usage of the tracing system

The Framework and all other M5T products use the tracing system to ease debugging and support. All products use the trace system in the same way.

- All traces use trace nodes specific to the product:
 - The Framework uses the trace nodes detailed in "g_stFramework and descendant tracing nodes" (see page 7)
- Trace levels are used as follows:

Level	Type
0	Unused (Reserved for future use)
1	Critical situations (imminent crash)
2	Error situations (failed operation)
3	Unused (Reserved for future use)
4	Basic Debugging information
5	Unused (Reserved for future use)
6	Debug Method/Function Entry
7	Debug Method/Function Exit
8	Extra Debugging information
9	Unused (Reserved for future use)

Using Framework Tracing Mechanism in Your Application

If you would like to use the framework tracing mechanism to output traces from your own application, you need to decide how to categorize your traces, for example by package, by folder, by module, or by any other criterion, then create your tracing hierarchy and attach it to the root node. This way, you have full control on your customized tracing mechanism.

2.2 - Assertion Mechanism Overview

The framework offers a complete assertion mechanism solution with different levels and extensions. Assertions are used in debug mode to track down problems related to limit conditions, unhandled conditions, or so-called impossible conditions.

To enable the assertions in an application, one of the MXD_ASSERT_ENABLE_STD (see page 272) macros must be defined in the application's build.

The assertion mechanism is highly configurable and scalable. The behaviour on assertion failure can be changed by the user at compile time or at run time and different assertions macros can be enabled or disabled at compile time to add or remove validations.

The assertion mechanism makes use of various handlers when an assertion fails:

- The "Assertion Failed Handler" is the first handler called when an assertion fails. When assertions are enabled, the default handler calls the trace handler, the call-stack trace handler, and the final behaviour handler.
- The "Trace Handler", called by the previous handler, prints or sends a trace about the failed assertion. The information outputted contains the assertion condition along with the file name and line number where the assertion occurred as in the following example:
"Assertion Failed (true == false) [152]SomeFile.cpp"
- The "Call-Stack Trace Handler", also called by the first handler, outputs the current state of the call-stack.
- Finally, the "Final Behaviour Handler" is called last by the first handler, after all tracing handlers have been called to terminate or restart the application.

All of these handlers can be overridden or disabled by defining the proper compilation #define.

By default, the assertion mechanism uses the tracing mechanism. To do otherwise, the tracing mechanism or the assertion behaviour must be overridden using the MXD_ASSERT_TRACE_OVERRIDE (see page 274) macro or by setting a new handler by calling MxAssertSetNewTraceHandler (see page 20) at run time. If traces are not enabled and the tracing handler is not overridden, the application may look like it just hung without any indication.

The following assertion macros are available within the framework. Note that if an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

MX_ASSERT

Basic assertion macro. Takes only one parameter: the assertion itself.

```
MX_ASSERT(i < m_vecItems.Size());
```

MX_ASSERT_EX

Basic assertion macro. Other than the assertion, takes an extra comment to clarify the assertion. If the assertion fails, the comment is included in the assertion trace.

```
MX_ASSERT_EX(i < m_vecItems.Size(), "Illegal access on m_vecItems.");
```

MX_ASSERT_RT

Same as MX_ASSERT (see page 39) but with a real-time configuration level. This assertion is used in time critical code so it can be added or removed easily to avoid adding latency to the execution.

```
MX_ASSERT_RT(m_rtpPacket.IsValid());
```

MX_ASSERT_RT_EX

Same as MX_ASSERT_RT (see page 42) but with an extra comment to clarify the assertion.

```
MX_ASSERT_RT_EX(m_rtpPacket.IsValid(), "Invalid RTP packet!");
```

MX_ASSERT_PERROR

Asserts that `errno` is zero (no error). Prints `errno` if the assertion fails.

```
MX_ASSERT_PERROR();
```

MX_ASSERT_PERROR_EX

Same as MX_ASSERT_PERROR (see page 41) but with an extra comment to clarify the assertion.

```
MX_ASSERT_PERROR_EX("Call to bind() failed.");
```

MX_ASSERT_PERROR_RT

Same as MX_ASSERT_RT (see page 42) but asserts on `errno`.

```
MX_ASSERT_PERROR_RT();
```

MX_ASSERT_PERROR_RT_EX

Same as MX_ASSERT_RT_EX (see page 42) but asserts on `errno`.

```
MX_ASSERT_PERROR_RT_EX("send() failed!");
```

MX_ASSERT_ONLY

Used to declare variables used in assertion macros only. This is to avoid annoying compiler warnings for unused variables.

```
MX_ASSERT_ONLY(bool bValid = false);
MX_ASSERT(bValid);
```

Notes

All the assertion macros involving `errno` described above should be used with care since not all functions set it. Moreover, some functions set `errno` only on a specific operating system.

2.3 - M5T Framework tracing nodes

This section documents all tracing nodes that are available in the M5T Framework. See the 'Tracing Nodes' subsection in General Tracing Configuration (see page 2) for more details.

2.3.1 - g_stTraceRoot tracing node

This is the root node of the tracing mechanism.

C++

```
STraceNode g_stTraceRoot;
```

Description

This node always exists and it corresponds to the root of the tree containing all the tracing nodes. It is registered by default and cannot be unregistered.

It is possible to enable or disable the root node recursively if it is desired to change the state of all the nodes in the tree at the same time. In this case, reference the root node by using "/" as in the following example.

See Also

[MxTraceRegisterNode](#) (see page 30), [MxTraceUnregisterNode](#) (see page 34), [MxTraceEnableNode](#) (see page 27), [MxTraceDisableNode](#) (see page 24), [General Tracing Configuration](#) (see page 2)

Example

```
// Enable all the nodes in the tree at the same time.
MxTraceEnableNode("/", true);
```

2.3.2 - g_stFramework and descendant tracing nodes

Framework package tracing nodes.

C++

```
STraceNode g_stFramework;
STraceNode g_stFrameworkBasic;
STraceNode g_stFrameworkCap;
STraceNode g_stFrameworkCrypto;
STraceNode g_stFrameworkECom;
STraceNode g_stFrameworkEComCEComUnknown;
STraceNode g_stFrameworkKerberos;
STraceNode g_stFrameworkKernel;
STraceNode g_stFrameworkKernelCFile;
STraceNode g_stFrameworkMocanaSs;
STraceNode g_stFrameworkNetwork;
STraceNode g_stFrameworkNetworkCAsyncSocketFactory;
STraceNode g_stFrameworkNetworkCAsyncTcpServerSocket;
STraceNode g_stFrameworkNetworkCAsyncTcpSocket;
STraceNode g_stFrameworkNetworkCAsyncUdpSocket;
STraceNode g_stFrameworkNetworkCPollRequestStatus;
STraceNode g_stFrameworkNetworkCPollRequestStatusPoll;
STraceNode g_stFrameworkNetworkCPollSocket;
STraceNode g_stFrameworkNetworkCPollSocketPoll;
STraceNode g_stFrameworkNetworkCTcpServerSocket;
STraceNode g_stFrameworkNetworkCTcpSocket;
STraceNode g_stFrameworkNetworkCTcpSocketSendRecv;
STraceNode g_stFrameworkNetworkCUdpSocket;
STraceNode g_stFrameworkNetworkSocketErrors;
STraceNode g_stFrameworkPki;
STraceNode g_stFrameworkRegExp;
STraceNode g_stFrameworkResolver;
STraceNode g_stFrameworkServicingThread;
STraceNode g_stFrameworkServicingThreadCEventDriven;
STraceNode g_stFrameworkServicingThreadCServicingThread;
STraceNode g_stFrameworkServicingThreadCServicingThreadActivate;
STraceNode g_stFrameworkServicingThreadCServicingThreadMessageService;
STraceNode g_stFrameworkServicingThreadCServicingThreadSocketService;
STraceNode g_stFrameworkServicingThreadCServicingThreadTimerService;
STraceNode g_stFrameworkTime;
STraceNode g_stFrameworkTls;
STraceNode g_stFrameworkTlsCAsyncTlsServerSocket;
STraceNode g_stFrameworkTlsCAsyncTlsServerSocketBase;
STraceNode g_stFrameworkTlsCAsyncTlsSocket;
STraceNode g_stFrameworkTlsCAsyncTlsSocketBase;
STraceNode g_stFrameworkTlsCTlsSocketSendRecv;
STraceNode g_stFrameworkTlsCAsyncTlsSocketFactoryCreationMgr;
STraceNode g_stFrameworkTlsCTlsContext;
STraceNode g_stFrameworkTlsCTlsSession;
STraceNode g_stFrameworkTlsCTlsSessionOpenSsl;
STraceNode g_stFrameworkTlsCTlsSessionMocanaSs;
STraceNode g_stFrameworkXml;
STraceNode g_stFrameworkXmlParserExpat;
```

```

STraceNode g_stFrameworkXmlParserExpatEcom;
STraceNode g_stFrameworkXmlGenericWriter;
STraceNode g_stFrameworkXmlGenericWriterEcom;
STraceNode g_stFrameworkXmlDocument;
STraceNode g_stFrameworkXmlDocumentEcom;
STraceNode g_stFrameworkXmlElement;

```

Description

These are the Framework tracing nodes. They are used for all the traces originating from the Framework, and MUST not be used for traces from other packages. To use these tracing nodes, you must include Config/MxConfig.h file.

Trace Node	Description
g_stFramework	Root node of the M5T Framework package.
g_stFrameworkBasic	Node of the 'Basic' folder.
g_stFrameworkCap	Node of the 'Cap' folder.
g_stFrameworkCrypto	Node of the 'Crypto' folder.
g_stFrameworkECom	Node of the 'ECom' folder.
g_stFrameworkEComCEComUnknown	Node specific to the 'CEComUnknown' (see page 414) class.
g_stFrameworkKerberos	Node of the 'Kerberos' folder.
g_stFrameworkKernel	Node of the 'Kernel' folder.
g_stFrameworkKernelCFile	Node specific to the 'CFile' (see page 472) class.
g_stFrameworkMocanaSs	Node for the MocanaSs library.
g_stFrameworkNetwork	Node of the 'Network' folder.
g_stFrameworkNetworkCAsyncSocketFactory	Node specific to the 'CAsyncSocketFactory' (see page 523) class.
g_stFrameworkNetworkCAsyncTcpServerSocket	Node specific to the 'CAsyncTcpServerSocket' class.
g_stFrameworkNetworkCAsyncTcpSocket	Node specific to the 'CAsyncTcpSocket' class.
g_stFrameworkNetworkCAsyncUdpSocket	Node specific to the 'CAsyncUdpSocket' class.
g_stFrameworkNetworkCPollRequestStatus	Node specific to the 'CPollRequestStatus' (see page 536) class.
g_stFrameworkNetworkCPollRequestStatusPoll	Node specific to the 'Poll' part of the 'CPollRequestStatus' (see page 536) class.
g_stFrameworkNetworkCPollSocket	Node specific to the 'CPollSocket' (see page 540) class.
g_stFrameworkNetworkCPollSocketPoll	Node specific to the 'Poll' part of the 'CPollSocket' (see page 540) class.
g_stFrameworkNetworkCTcpServerSocket	Node specific to the 'CTcpServerSocket' (see page 559) class.
g_stFrameworkNetworkCTcpSocket	Node specific to the 'CTcpSocket' (see page 567) class.
g_stFrameworkNetworkCTcpSocketSendRecv	Node specific to the 'Send/Receive' parts of the 'CTcpSocket' (see page 567) class.
g_stFrameworkNetworkCUdpSocket	Node specific to the 'CUdpSocket' (see page 585) class.
g_stFrameworkNetworkSocketErrors	Node specific to the 'SocketErrors' file.
g_stFrameworkPki	Node of the 'Pki' folder.
g_stFrameworkRegExp	Node of the 'RegExp' folder.
g_stFrameworkResolver	Node of the 'Resolver' folder.
g_stFrameworkServicingThread	Node of the 'ServicingThread' folder.
g_stFrameworkServicingThreadCEventDriven	Node specific to the 'CEventDriven' (see page 762) class.
g_stFrameworkServicingThreadCServicingThread	Node specific to the 'CServicingThread' (see page 771) class.
g_stFrameworkServicingThreadCServicingThreadActivate	Node specific to the 'Activation mechanism' of the 'CServicingThread' (see page 771) class.

g_stFrameworkServicingThreadCServicingThreadMessageService	Node specific to the 'Message' service of the 'CServicingThread' (see page 771) class.
g_stFrameworkServicingThreadCServicingThreadSocketService	Node specific to the 'Socket' service of the 'CServicingThread' (see page 771) class.
g_stFrameworkServicingThreadCServicingThreadTimerService	Node specific to the 'Timer' service of the 'CServicingThread' (see page 771) class.
g_stFrameworkTime	Node of the 'Time' folder.
g_stFrameworkTls	Node of the 'Tls' folder.
g_stFrameworkTlsCAsyncTlsServerSocket	Node specific to the 'CAsyncTlsServerSocket' class.
g_stFrameworkTlsCAsyncTlsServerSocketBase	Node specific to the 'CAsyncTlsServerSocketBase' class.
g_stFrameworkTlsCAsyncTlsSocket	Node specific to the 'CAsyncTlsSocket' class.
g_stFrameworkTlsCAsyncTlsSocketBase	Node specific to the 'CAsyncTlsSocketBase' class.
g_stFrameworkTlsCAsyncTlsSocketFactoryCreationMgr	Node specific to the 'CAsyncTlsSocketFactoryCreationMgr' (see page 794) class.
g_stFrameworkTlsCTlsContext	Node specific to the 'CTlsContext' (see page 797) class.
g_stFrameworkTlsCTlsSession	Node specific to the 'CTlsSession' (see page 803) class.
g_stFrameworkTlsCTlsSessionOpenSsl	Node specific to the 'CTlsSessionOpenSsl' class.
g_stFrameworkTlsCTlsSessionMocanaSs	Node specific to the 'CTlsSessionMocanaSs' class.
g_stFrameworkXml	Node of the 'Xml' folder.

Trace Node	String to access the node
g_stFramework	/Framework
g_stFrameworkBasic	/Framework/Basic
g_stFrameworkCap	/Framework/Cap
g_stFrameworkCrypto	/Framework/Crypto
g_stFrameworkECom	/Framework/ECom
g_stFrameworkEComCEComUnknown	/Framework/ECom/CEComUnknown
g_stFrameworkKerberos	/Framework/Kerberos
g_stFrameworkKernel	/Framework/Kernel
g_stFrameworkKernelCFile	/Framework/Kernel/CFile
g_stFrameworkMocanaSs	/Framework/MocanaSs
g_stFrameworkNetwork	/Framework/Network
g_stFrameworkNetworkCAsyncSocketFactory	/Framework/Network/CAsyncSocketFactory
g_stFrameworkNetworkCAsyncTcpServerSocket	/Framework/Network/CAsyncTcpServerSocket
g_stFrameworkNetworkCAsyncTcpSocket	/Framework/Network/CAsyncTcpSocket
g_stFrameworkNetworkCAsyncUdpSocket	/Framework/Network/CAsyncUdpSocket
g_stFrameworkNetworkCPollRequestStatus	/Framework/Network/CPollRequestStatus
g_stFrameworkNetworkCPollRequestStatusPoll	/Framework/Network/CPollRequestStatus/Poll
g_stFrameworkNetworkCPollSocket	/Framework/Network/CPollSocket
g_stFrameworkNetworkCPollSocketPoll	/Framework/Network/CPollSocket/Poll
g_stFrameworkNetworkCTcpServerSocket	/Framework/Network/CTcpServerSocket
g_stFrameworkNetworkCTcpSocket	/Framework/Network/CTcpSocket
g_stFrameworkNetworkCTcpSocketSendRecv	/Framework/Network/CTcpSocket/SendRecv

g_stFrameworkNetworkCUdpSocket	/Framework/Network/CUdpSocket
g_stFrameworkNetworkSocketErrors	/Framework/Network/SocketErrors
g_stFrameworkPki	/Framework/Pki
g_stFrameworkRegExp	/Framework/RegExp
g_stFrameworkResolver	/Framework/Resolver
g_stFrameworkServicingThread	/Framework/ServicingThread
g_stFrameworkServicingThreadCEventDriven	/Framework/ServicingThread/CEventDriven
g_stFrameworkServicingThreadCServicingThread	/Framework/ServicingThread/CServicingThread
g_stFrameworkServicingThreadCServicingThreadActivate	/Framework/ServicingThread/CServicingThread/Activate
g_stFrameworkServicingThreadCServicingThreadMessageService	/Framework/ServicingThread/CServicingThread/MessageService
g_stFrameworkServicingThreadCServicingThreadSocketService	/Framework/ServicingThread/CServicingThread/SocketService
g_stFrameworkServicingThreadCServicingThreadTimerService	/Framework/ServicingThread/CServicingThread/TimerService
g_stFrameworkTime	/Framework/Time
g_stFrameworkTls	/Framework/Tls
g_stFrameworkTlsCAsyncTlsServerSocket	/Framework/Tls/CAsyncTlsServerSocket
g_stFrameworkTlsCAsyncTlsServerSocketBase	/Framework/Tls/CAsyncTlsServerSocketBase
g_stFrameworkTlsCAsyncTlsSocket	/Framework/Tls/CAsyncTlsSocket
g_stFrameworkTlsCAsyncTlsSocketBase	/Framework/Tls/CAsyncTlsSocketBase
g_stFrameworkTlsCAsyncTlsSocketFactoryCreationMgr	/Framework/Tls/CAsyncTlsSocketFactoryCreationMgr
g_stFrameworkTlsCTlsContext	/Framework/Tls/CTlsContext
g_stFrameworkTlsCTlsSession	/Framework/Tls/CTlsSession
g_stFrameworkTlsCTlsSessionOpenSsl	/Framework/Tls/CTlsSessionOpenSsl
g_stFrameworkTlsCTlsSessionMocanaSs	/Framework/Tls/CTlsSessionMocanaSs
g_stFrameworkXml	/Framework/Xml

Location

Startup/CFrameworkInitializer.cpp

See Also

g_stTraceRoot (see page 7), MxTraceRegisterNode (see page 30), MxTraceUnregisterNode (see page 34), MxTraceEnableNode (see page 27), MxTraceDisableNode (see page 24), General Tracing Configuration (see page 2)

2.4 - Basic

This section documents the Sources/Basic folder of the M5T Framework. It is divided in functional subsections:

- Enumerations (see page 10)
- Functions (see page 17)
- Macros (see page 37)
- Deprecated Macros (see page 62)
- Structures (see page 62)
- Templates (see page 64)
- Types (see page 84)
- Variables (see page 94)

2.4.1 - Enumerations

This section documents the enumerations of the Sources/Basic folder.

Enumerations

Enumeration	Description
EEventNotifier (see page 11)	Events that can be observed.
EMxBase (see page 11)	Defines the possible bases that can be used to represent an integer.
EMxPackageId (see page 12)	Defines the possible packages for the Package ID of the mxt_result (see page 92) type.
EMxResultSharedFailCriticalCodeId (see page 13)	This enumeration defines the shared fail-critical code IDs.
EMxResultSharedFailErrorCodeId (see page 13)	This enumeration defines the shared fail-error code IDs.
EMxResultSharedSuccessInfoCodeId (see page 14)	Defines the shared success-info IDs.
EMxResultSharedSuccessWarningCodeId (see page 15)	This enumeration defines the shared success-warning code IDs.
EMxTraceField (see page 15)	Defines the possible trace fields.
EMxTraceLevel (see page 16)	Defines the possible trace levels.

2.4.1.1 - EEventNotifier Enumeration

Events that can be observed.

C++

```
enum EEventNotifier {
    eEN_FQDN_NOT_RESOLVED,
    eEN_CMEMORYALLOCATOR_OUT_OF_MEMORY,
    eEN_CMEMORYALLOCATOR_TRACKING,
    eEN_LAST_EVENT
};
```

Description

Enumeration of events that can be observed by an external observer.

Members

Members	Description
eEN_FQDN_NOT_RESOLVED	This event is generated when the network abstraction layer is unable to resolve the target FQDN name. The parameter to this event is the FQDN that was not resolved: "... = const char* pszFqdn"
eEN_CMEMORYALLOCATOR_OUT_OF_MEMORY	This event is generated when CMemoryAllocator (see page 480) is unable to allocate a block of memory. The parameter is set to NULL.
eEN_CMEMORYALLOCATOR_TRACKING	This event is generated when CMemoryAllocator (see page 480) reaches a certain amount of allocated memory. The parameter is set to the value of the allocated memory, in bytes.
eEN_LAST_EVENT	This is the last event. This is never generated and is kept only as an index.

2.4.1.2 - EMxBase Enumeration

Defines the possible bases that can be used to represent an integer.

C++

```
typedef enum {
    eBASE_AUTO = 0,
    eBASE_BINARY = 2,
    eBASE_OCTAL = 8,
    eBASE_DECIMAL = 10,
    eBASE_HEXADECIMAL = 16,
    eBASE_INVALID
} EMxBase;
```

Description

This enumeration defines the bases that can be used to represent an integer.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

Members

Members	Description
eBASE_AUTO = 0	Base determined automatically. Only used in CSocketAddr (see page 545).
eBASE_BINARY = 2	Binary base.

eBASE_OCTAL = 8	Octal base.
eBASE_DECIMAL = 10	Decimal base.
eBASE_HEXADECIMAL = 16	Hexadecimal base.
eBASE_INVALID	Invalid base.

2.4.1.3 - EMxPackageld Enumeration

Defines the possible packages for the Package ID of the mxt_result (see page 92) type.

C++

```
typedef enum {
    eMX_PKG_NONE = 0,
    eMX_PKG_ASSERTION = 1,
    eMX_PKG_MITOSFW,
    eMX_PKG_RTP,
    eMX_PKG_SRTP,
    eMX_PKG_SIP,
    eMX_PKG_SIP_PARSER,
    eMX_PKG_SIPTRANSPORT,
    eMX_PKG_SIPTRANSACTION,
    eMX_PKG_STPCORE,
    eMX_PKG_SIPCORESVC,
    eMX_PKG_SIPUSERAGENT,
    eMX_PKG_SIP_PROXY,
    eMX_PKG_MIX,
    eMX_PKG_MIX_SNMP_AGENT_ADAPTOR,
    eMX_PKG_MIX_PERSISTENCE_MANAGER,
    eMX_PKG_STUN,
    eMX_PKG_TFW,
    eMX_PKG_MIKEY,
    eMX_PKG_STGCOMP,
    eMX_PKG_SIPUACOMPONENTS,
    eMX_PKG_SIPSERVERCOMPONENTS,
    eMX_PKG_SDP,
    eMX_PKG_MTEI,
    eMX_PKG_COMMAND_DIGITS_INTERFACE,
    eMX_PKG_FXSCALLMANAGER,
    eMX_PKG_UASSPINTEGRATION,
    eMX_PKG_UASSP,
    eMX_PKG_TR_111_PART_2_SERVER,
    eMX_PKG_ZRTP,
    eMX_PKG_HTTP,
    eMX_PKG_PSEM,
    eMX_PKG_ICE,
    eMX_PKG_SIZE,
    eMX_PKG_ALL = eMX_PKG_SIZE
} EMxPackageld;
```

Description

This enumeration defines the possible packages with result values throughout the M5T offered products.

The values in this enumeration can not form a bit mask, but are instead sequential values.

The package IDs are meant to be augmented by the user. The application must define MXD_PKG_ID_OVERRIDE (see page 300) if it wants to reuse the result mechanism for its own functions/methods.

Location

Defined in Basic/PkgId.h but must include Config/MxConfig.h to access this.

See Also

MXD_PKG_ID_OVERRIDE (see page 300)

Members

Members	Description
eMX_PKG_NONE = 0	First package ID. Reserved for future use.
eMX_PKG_ASSERTION = 1	Package ID for the assertion mechanism.
eMX_PKG_MITOSFW	Package ID for the Framework package.
eMX_PKG_RTP	Package ID for the RTP package.
eMX_PKG_SRTP	Package ID for the SRTP package.
eMX_PKG_SIP	Package ID for the SIP package.
eMX_PKG_SIP_PARSER	Package ID for the SIP parser package.
eMX_PKG_SIPTRANSPORT	Package ID for the SIP transport package.

eMX_PKG_SIPTRANSACTION	Package ID for the SIP transaction package.
eMX_PKG_SIPCORE	Package ID for the SIP core package.
eMX_PKG_SIPCORESVC	Package ID for the SIP core SVC package.
eMX_PKG_SIPUSERAGENT	Package ID for the SIP user agent package.
eMX_PKG_SIP_PROXY	Package ID for the SIP proxy package.
eMX_PKG_MIX	Package ID for the MIX package.
eMX_PKG_MIX_SNMP_AGENT_ADAPTER	Package ID for the MIX SNMP agent adaptor package.
eMX_PKG_MIX_PERSISTENCE_MANAGER	Package ID for the MIX persistence manager package.
eMX_PKG_STUN	Package ID for the STUN package.
eMX_PKG_TFW	Package ID for the TestFramework package.
eMX_PKG_MIKEY	Package ID for the Mikey package.
eMX_PKG_SIGCOMP	Package ID for the Sigcomp package.
eMX_PKG_SIPUACOMPONENTS	Package ID for the SIP UA components package.
eMX_PKG_SIPSERVERCOMPONENTS	Package ID for the SIP server components package.
eMX_PKG_SDP	Package ID for the SDP package.
eMX_PKG_MTEI	Package ID for the MTEI.
eMX_PKG_COMMAND_DIGITS_INTERFACE	Package ID for the Command Digits Interface.
eMX_PKG_FXSCALLMANAGER	Package ID for the FXS Callmanager.
eMX_PKG_UASSPINTTEGRATION	Package ID for the UA SSP Integration.
eMX_PKG_UASSP	Package ID for the UA SSP.
eMX_PKG_TR_111_PART_2_SERVER	Package ID for the TR-111 part 2 server.
eMX_PKG_ZRTP	Package ID for the ZRTP package.
eMX_PKG_HTTP	Package ID for the HTTP package.
eMX_PKG_PSEM	Package ID for the PSEM package.
eMX_PKG_ICE	Package ID for the ICE package.
eMX_PKG_SIZE	Indicates the size of the EMxPackageId enum.
eMX_PKG_ALL = eMX_PKG_SIZE	Package ID representing all packages.

2.4.1.4 - EMxResultSharedFailCriticalCodeId Enumeration

This enumeration defines the shared fail-critical code IDs.

C++

```
enum EMxResultSharedFailCriticalCodeId {
    resFC_FIRST = (int)(MX_RESULT_LEVEL_CRITICAL_VALUE | 1),
    resFC_CRITICAL = resFC_FIRST,
    resFC_LAST = resFC_CRITICAL
};
```

Description

This enumeration defines the shared fail-critical code IDs.

The fail-critical codes are meant to carry additional information when a function/method encounters an unhandled condition that can leave the system unstable or the execution can not resume without possible consequences. It is also used to notify the caller of unrecoverable conditions.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

[mxt_result](#) (see page 92)

Members

Members	Description
resFC_FIRST = (int)(MX_RESULT_LEVEL_CRITICAL_VALUE 1)	First fail-critical result code. Used as an index only.
resFC_CRITICAL = resFC_FIRST	Generic critical failure.
resFC_LAST = resFC_CRITICAL	Last fail-critical result code. Used as an index only.

2.4.1.5 - EMxResultSharedFailErrorCodeId Enumeration

This enumeration defines the shared fail-error code IDs.

C++

```
enum EMxResultSharedFailErrorCodeId {
```

```

resFE_FIRST = (int)(MX_RESULT_LEVEL_ERROR_VALUE | 1),
resFE_FAIL = resFE_FIRST,
resFE_INVALID_STATE,
resFE_INVALID_ARGUMENT,
resFE_NOT_IMPLEMENTED,
resFE_NULL_POINTER,
resFE_UNEXPECTED,
resFE_OUT_OF_MEMORY,
resFE_ACCESS_DENIED,
resFE_DUPLICATE,
resFE_ABORT,
resFE_TIMEOUT,
resFE_NOT_FOUND,
resFE_LAST = resFE_NOT_FOUND
};

```

Description

This enumeration defines the shared fail-error code IDs.

The fail-error codes are meant to carry additional information when a function/method fails or encounters an unhandled condition. If no additional information is to be retuned, it is recommended to use the shared resFE_FAIL return code ID.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

[mxt_result](#) (see page 92)

Members

Members	Description
resFE_FIRST = (int)(MX_RESULT_LEVEL_ERROR_VALUE 1)	First fail-error result code. Used as an index only.
resFE_FAIL = resFE_FIRST	Fail-error code indicating a failure.
resFE_INVALID_STATE	Fail-error code result indicating an invalid state.
resFE_INVALID_ARGUMENT	Fail-error code result indicating an invalid argument.
resFE_NOT_IMPLEMENTED	Fail-error code result indicating a non-implemented feature.
resFE_NULL_POINTER	Fail-error code result indicating a NULL pointer.
resFE_UNEXPECTED	Fail-error code result indicating an unexpected error.
resFE_OUT_OF_MEMORY	Fail-error code result indicating an out of memory error.
resFE_ACCESS_DENIED	Fail-error code result indicating an access denied error.
resFE_DUPLICATE	Fail-error code result indicating a duplicate error.
resFE_ABORT	Fail-error code result indicating an abort error.
resFE_TIMEOUT	Fail-error code result indicating a timeout condition.
resFE_NOT_FOUND	Fail-error code result indicating an element not found error.
resFE_LAST = resFE_NOT_FOUND	Last fail-error result code. Used as an index only.

2.4.1.6 - EMxResultSharedSuccessInfoCodeId Enumeration

Defines the shared success-info IDs.

C++

```

enum EMxResultSharedSuccessInfoCodeId {
    resSI_FIRST = MX_RESULT_LEVEL_INFO_VALUE,
    resS_OK = resSI_FIRST,
    resSI_TRUE,
    resSI_FALSE,
    resSI_LAST = resSI_FALSE
};

```

Description

This enumeration defines the shared success-info code IDs.

The first value has to be resS_OK. The success-information codes are meant to carry additional information when a function/method succeeds. It is recommended to use the shared resS_OK result when no additional information needs to be returned.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

[mxt_result](#) (see page 92)

Members

Members	Description
resSI_FIRST = MX_RESULT_LEVEL_INFO_VALUE	LL = 00, PkgId = 0, Code ID = 0
resS_OK = resSI_FIRST	Success-info code result indicating sucessfull completion.
resSI_TRUE	Success-info code result indicating a TRUE value.
resSI_FALSE	Success-info code result indicating a FALSE value.
resSI_LAST = resSI_FALSE	Last success-info result code. Used as an index only.

2.4.1.7 - EMxResultSharedSuccessWarningCodeId Enumeration

This enumeration defines the shared success-warning code IDs.

C++

```
enum EMxResultSharedSuccessWarningCodeId {
    resSW_FIRST = (MX_RESULT_LEVEL_WARNING_VALUE | 1),
    resSW_WARNING = resSW_FIRST,
    resSW_NOTHING_DONE,
    resSW_ASYNC_PROCESSING,
    resSW_LAST = resSW_ASYNC_PROCESSING
};
```

Description

This enumeration defines the shared success-warning code IDs.

The success-warning codes are meant to carry additional information when a function/method succeeds, but with the addition of raising a yellow flag.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

[mxt_result](#) (see page 92)

Members

Members	Description
resSW_FIRST = (MX_RESULT_LEVEL_WARNING_VALUE 1)	First success-warning result code. Used as an index only.
resSW_WARNING = resSW_FIRST	Generic warning.
resSW_NOTHING_DONE	No operation has been performed.
resSW_ASYNC_PROCESSING	An asynchronous operation is in progress.
resSW_LAST = resSW_ASYNC_PROCESSING	Last success-warning result code. Used as an index only.

2.4.1.8 - EMxTraceField Enumeration

Defines the possible trace fields.

C++

```
typedef enum {
    eMXTFIELD_PRI,
    eMXTFIELD_DATE_TIME,
    eMXTFIELD_HOSTNAME,
    eMXTFIELD_TRACINGLEVEL,
    eMXTFIELD_PROGNAME,
    eMXTFIELD_PROCESSID,
    eMXTFIELD_THREADID,
    eMXTFIELD_TIME_STAMP,
    eMXTFIELD_SEQUENCE_NO,
    eMXTFIELD_LAST_FIELD
} EMxTraceField;
```

Description

This enumeration defines the possible fields associated with each trace. The fields are predefined and are not meant to be augmented by the application. The fields can be enabled and disabled at run time with the functions [MxTraceEnableField](#) (see page 26) and [MxTraceDisableField](#) (see page 24).

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceEnableField](#) (see page 26), [MxTraceDisableField](#) (see page 24), [MxTraceEnableAllFields](#) (see page 25)

Members

Members	Description
eMXTFIELD_PRI	Syslog Type Level field.
eMXTFIELD_DATE_TIME	Syslog Date and Time field.
eMXTFIELD_HOSTNAME	Hostname field.
eMXTFIELD_TRACINGLEVEL	Tracing level.
eMXTFIELD_PROGNAME	Program name field.
eMXTFIELD_PROCESSID	Process ID field.
eMXTFIELD_THREADID	Thread name field.
eMXTFIELD_TIME_STAMP	Timestamp field.
eMXTFIELD_SEQUENCE_NO	Sequence number field.
eMXTFIELD_LAST_FIELD	Last field. Used as an index only.

2.4.1.9 - EMxTraceLevel Enumeration

Defines the possible trace levels.

C++

```
typedef enum {
    eLEVEL_NONE = 0x00000000,
    eLEVEL0 = 0x00000001,
    eLEVEL1 = 0x00000002,
    eLEVEL2 = 0x00000004,
    eLEVEL3 = 0x00000008,
    eLEVEL4 = 0x00000010,
    eLEVEL5 = 0x00000020,
    eLEVEL6 = 0x00000040,
    eLEVEL7 = 0x00000080,
    eLEVEL8 = 0x00000100,
    eLEVEL9 = 0x00000200,
    eLEVEL_ALL = (int)0xFFFFFFFF
} EMxTraceLevel;
```

Description

This enumeration defines the possible levels associated with each trace. The levels are predefined and are not meant to be augmented by the application.

Of course, a trace cannot belong to the level eLEVEL_NONE.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[General tracing configuration](#) (see page 2), [MXD_TRACE0_ENABLE_SUPPORT](#) (see page 314), [MxTraceEnableNodeMaxLevels](#) (see page 28), [MxTraceEnableNodeLevel](#) (see page 27)

Members

Members	Description
eLEVEL_NONE = 0x00000000	None level. Should never be used.
eLEVEL0 = 0x00000001	Tracing level 0.
eLEVEL1 = 0x00000002	Tracing level 1.
eLEVEL2 = 0x00000004	Tracing level 2.
eLEVEL3 = 0x00000008	Tracing level 3.
eLEVEL4 = 0x00000010	Tracing level 4.
eLEVEL5 = 0x00000020	Tracing level 5.
eLEVEL6 = 0x00000040	Tracing level 6.
eLEVEL7 = 0x00000080	Tracing level 7.
eLEVEL8 = 0x00000100	Tracing level 8.
eLEVEL9 = 0x00000200	Tracing level 9.
eLEVEL_ALL = (int)0xFFFFFFFF	Tracing level ALL. Used to specify all levels at once.

2.4.2 - Functions

This section documents the functions of the Sources/Basic folder.

Functions

Function	Description
AsciiToHex (see page 18)	Converts an ASCII value to its hexadecimal value in the ASCII chart.
HexToAscii (see page 18)	Converts a hexadecimal value to its ASCII value in the ASCII chart.
InstallGenericFaultHandler (see page 19)	Installs fault handler.
MxAssertSetNewCallStackTraceHandler (see page 19)	Installs a new call stack trace handler.
MxAssertSetNewFailHandler (see page 19)	Installs a new failure handler.
MxAssertSetNewFinalBehaviorHandler (see page 19)	Installs a new final behavior handler.
MxAssertSetNewTraceHandler (see page 20)	Installs a new assert trace handler.
MxIntToString (see page 20)	Converts a signed integer into a character string, in the given 'base' representation.
MxSnprintf (see page 21)	Writes into a buffer a formatted character string.
MxStrErrorReentrant (see page 21)	Copies the system error message into the given buffer.
MxStringCaseCompare (see page 21)	Performs a comparison between two character strings. Case-insensitive string comparison.
MxStringToInt (see page 22)	Converts a null-terminated character string into a signed integer, in the given 'base' representation.
MxStringToInt (see page 22)	Converts a character buffer into a signed integer, in the given 'base' representation.
MxStringToInt (see page 22)	Converts a null-terminated character string into an unsigned integer, in the given 'base' representation.
MxStringToUInt (see page 23)	Converts a character buffer into an unsigned integer, in the given 'base' representation.
MxTraceAddOutputHandler (see page 23)	Appends the given handler to the list of trace output handlers.
MxTraceDisableAllFields (see page 24)	Disables all tracing fields.
MxTraceDisableField (see page 24)	Disables a tracing field
MxTraceDisableLevel (see page 24)	Removes a tracing level.
MxTraceDisableNode (see page 24)	Disables a tracing node.
MxTraceDisableNodeLevel (see page 25)	Removes a tracing level for this node.
MxTraceEnableAllFields (see page 25)	Enables all tracing fields.
MxTraceEnableDefaultFields (see page 26)	Enables default tracing fields.
MxTraceEnableField (see page 26)	Enables a tracing field
MxTraceEnableLevel (see page 26)	Adds a tracing level.
MxTraceEnableMaxLevels (see page 26)	Adds all tracing levels less than or equal to the level passed.
MxTraceEnableNode (see page 27)	Enables a tracing node.
MxTraceEnableNodeLevel (see page 27)	Adds a tracing level for this node.
MxTraceEnableNodeMaxLevels (see page 28)	Adds all tracing levels less than or equal to the level passed for this node.
MxTraceEnableSyslogFields (see page 28)	Enables syslog tracing fields.
MxTraceFieldIsEnabled (see page 29)	Returns true if a field is active, false if it is not.
MxTraceGetLevelsEnabled (see page 29)	Gets the tracing levels currently enabled.
MxTraceGetNodeLevelsEnabled (see page 29)	Gets the tracing levels currently enabled for this node.
MxTracePrintTree (see page 30)	Prints all enabled trace nodes of the tree.
MxTraceRegisterNode (see page 30)	Registers a tracing node.
MxTraceRemoveOutputHandler (see page 31)	Removes each occurrence of the given function pointer in the list of registered trace output handlers.
MxTraceSetHostNameField (see page 31)	Sets the local hostname field.
MxTraceSetNewCallStackHandler (see page 31)	Configures a new call stack trace handler.
MxTraceSetNewFormatHandler (see page 32)	Configures a new format handler.
MxTraceSetNewOutputHandler (see page 32)	Sets a new output handler to replace the previous ones.
MxTraceSetSysTimeFormatHandler (see page 32)	Configures a new system time format handler.
MxTraceSetTimeFormatHandler (see page 33)	Configures a new time stamp format handler.
MxTraceToDebugger (see page 33)	mxt_PFNTraceOutputHandler (see page 92) for writing traces to a debugger.
MxTraceToStderr (see page 33)	mxt_PFNTraceOutputHandler (see page 92) for writing traces to the standard error.
MxTraceToStdout (see page 34)	mxt_PFNTraceOutputHandler (see page 92) for writing traces to the standard output.
MxTraceUnregisterNode (see page 34)	Unregisters a tracing node.

MxUintToString (see page 35)	Converts an unsigned integer into a character string, in the given 'base' representation.
MxVsnprintf (see page 35)	Writes into a buffer a formatted character string.
NotifyEventObservers (see page 36)	Notifies observers.
RegisterEventObserver (see page 36)	Registers an observer.
UnInstallGenericFaultHandler (see page 36)	Uninstalls fault handler.
UnregisterEventObserver (see page 36)	Unregisters an observer.

2.4.2.1 - AsciiToHex Function

Converts an ASCII value to its hexadecimal value in the ASCII chart.

C++

```
unsigned int AsciiToHex(IN bool bConvertToUpper, IN const uint8_t* puAscii, IN unsigned int uAsciiSize, OUT
uint8_t* puHex, IN unsigned int uHexBufSize);
```

Parameters

Parameters	Description
IN bool bConvertToUpper	If true, hexadecimal alphabetical characters A-F are output in upper case.
IN const uint8_t* puAscii	Input data, valid ASCII characters in string format, values 0 through 255.
IN unsigned int uAsciiSize	Input data length to convert.
OUT uint8_t* puHex	Output buffer. Should have at least twice the length capacity of uAsciiSize.
IN unsigned int uHexBufSize	Output buffer capacity. It is not exceeded in output.

Returns

The length that has been written in the output buffer.

Description

Converts an ASCII value, from its string format, to its hexadecimal value in the ASCII chart, outputted in string format. See Examples.

See Also

HexToAscii (see page 18)

Example

```
"j" converts to "6A" (or "6a")
"ÿ" converts to "FF"
"123" converts to "313233"
```

2.4.2.2 - HexToAscii Function

Converts a hexadecimal value to its ASCII value in the ASCII chart.

C++

```
unsigned int HexToAscii(IN const uint8_t* puHex, IN unsigned int uHexSize, OUT uint8_t* puAscii, IN unsigned int
uAsciiBufSize);
```

Parameters

Parameters	Description
IN const uint8_t* puHex	Input data, hexadecimal characters in string format, 00 through FF. Upper and lower case values are equivalent.
IN unsigned int uHexSize	Length of the input buffer. It should be a multiple of 2.
OUT uint8_t* puAscii	Output buffer. It should have a capacity of at least half the uHexSize. The output is ASCII characters in the range 0-255.
IN unsigned int uAsciiBufSize	Output buffer capacity.

Returns

The length that has been written in the output buffer.

Description

Converts a hexadecimal value, from its string format, to its ASCII value in the ASCII chart, outputted in string format. See Examples.

See Also

AsciiToHex (see page 18)

Example

```
"6A" or "6a" converts to "j"
"FF" converts to "ÿ"
"313233" converts to "123"
```

2.4.2.3 - InstallGenericFaultHandler Function

Installs fault handler.

C++

```
bool InstallGenericFaultHandler();
```

Description

Under certain OSs and for a limited number of architectures, installs fault handlers. Called by Initialize if MXD_FAULT_HANDLER_ENABLE is defined. Otherwise, fault handlers are not installed if InstallGenericFaultHandler is not called.

Location

Defined in Basic/MxFault.h.

2.4.2.4 - MxAssertSetNewCallStackTraceHandler Function

Installs a new call stack trace handler.

C++

```
SAssertCallStackTraceHandler MxAssertSetNewCallStackTraceHandler(IN SAssertCallStackTraceHandler* pstNewHandler);
```

Parameters

Parameters	Description
IN SAssertCallStackTraceHandler* pstNewHandler	Pointer to the stack trace handler to install.

Returns

The previously installed stack trace handler function pointer.

Description

Installs a new handler to be used by the MX_ASSERT_XXX macros.

See Also

MxAssertSetNewFailHandler (see page 19), MxAssertSetNewTraceHandler (see page 20), MxAssertSetNewFinalBehaviorHandler (see page 19),

2.4.2.5 - MxAssertSetNewFailHandler Function

Installs a new failure handler.

C++

```
SAssertFailHandler MxAssertSetNewFailHandler(IN SAssertFailHandler* pstNewHandler);
```

Parameters

Parameters	Description
IN SAssertFailHandler* pstNewHandler	Pointer to the assert fail handler to install.

Returns

The previously installed assert fail handler function pointer.

Description

Installs a new handler to be used by the MX_ASSERT_XXX macros.

See Also

MxAssertSetNewTraceHandler (see page 20), MxAssertSetNewCallStackTraceHandler (see page 19), MxAssertSetNewFinalBehaviorHandler (see page 19)

2.4.2.6 - MxAssertSetNewFinalBehaviorHandler Function

Installs a new final behavior handler.

C++

```
SAssertFinalBehaviorHandler MxAssertSetNewFinalBehaviorHandler(IN SAssertFinalBehaviorHandler* pstNewHandler);
```

Parameters

Parameters	Description
IN SAssertFinalBehaviorHandler* pstNewHandler	Pointer to the final behaviour handler to install.

Returns

The previously installed final behaviour handler function pointer.

Description

Installs a new handler to be used by the MX_ASSERT_XXX macros.

See Also

[MxAssertSetNewFailHandler](#) (see page 19), [MxAssertSetNewTraceHandler](#) (see page 20), [MxAssertSetNewCallStackTraceHandler](#) (see page 19)

2.4.2.7 - MxAssertSetNewTraceHandler Function

Installs a new assert trace handler.

C++

```
SAssertTraceHandler MxAssertSetNewTraceHandler(IN SAssertTraceHandler* pstNewHandler);
```

Parameters

Parameters	Description
IN SAssertTraceHandler* pstNewHandler	Pointer to the trace handler to install.

Returns

The previously installed trace handler function pointer.

Description

Installs a new handler to be used by the MX_ASSERT_XXX macros.

See Also

[MxAssertSetNewFailHandler](#) (see page 19), [MxAssertSetNewCallStackTraceHandler](#) (see page 19), [MxAssertSetNewFinalBehaviorHandler](#) (see page 19)

2.4.2.8 - MxIntToString Function

Converts a signed integer into a character string, in the given 'base' representation.

C++

```
char* MxIntToString(IN int64_t nValue, IN unsigned int uCapacity, OUT char* pszBuffer, OUT unsigned int* puSize, IN EMxBase eBase = eBASE_DECIMAL);
```

Parameters

Parameters	Description
IN int64_t nValue	The signed integer to convert.
IN unsigned int uCapacity	Maximum number of characters that can be written into the output buffer (including the null-terminator).
OUT char* pszBuffer	Output buffer where to write the formatted character string.
OUT unsigned int* puSize	On exit, contains the number of characters written to pszBuffer, excluding the null-terminator.
IN EMxBase eBase = eBASE_DECIMAL	The base in which nValue is represented. Valid bases are eBASE_BINARY, eBASE_OCTAL, eBASE_DECIMAL, and eBASE_HEXADECIMAL.

Returns

On success, returns pszBuffer and puSize contains the number of characters written to pszBuffer, excluding the null-terminator. On failure, returns NULL.

Description

Converts a signed integer into a character string in the given 'base' representation.

2.4.2.9 - MxSnprintf Function

Writes into a buffer a formatted character string.

C++

```
int MxSnprintf(OUT char * pszBuffer, IN int nSize, IN const char * pszFormat, ...);
```

Parameters

Parameters	Description
OUT char * pszBuffer	Output buffer where to write the formatted character string.
IN int nSize	Maximum number of characters that can be written in the output buffer (including the null-terminator).
IN const char * pszFormat	A character string that specifies how subsequent arguments (variable number) are converted to the output buffer. ...: A variable number of arguments.

Returns

Returns the number of characters written to the buffer, excluding the null-terminator. If the buffer is too small, it writes as many characters as it can (i.e., nSize), without the null-terminator, and returns the value of the argument nSize.

Description

Writes a formatted character string into a buffer. This function is portable across all supported OSs and behaves identically except for the format (pszFormat), which is handled differently by each OS.

See Also

Linux manual page of vsnprintf(3), which differs only by the return value.

2.4.2.10 - MxStrErrorReentrant Function

Copies the system error message into the given buffer.

C++

```
int MxStrErrorReentrant(OUT char * pszBuffer, IN int nSize, IN int nErrorNumber);
```

Parameters

Parameters	Description
OUT char * pszBuffer	Output buffer where to write the formatted character string.
IN int nSize	Maximum number of characters that can be written into the output buffer (including the null-terminator).
IN int nErrorNumber	The system error number. On Linux and VxWorks, this is usually the value of errno. On Windows, it is the value of GetLastError, WSAGetLastError, or any other way a specific package may use.

Returns

Returns the number of characters written to the buffer, excluding the null-terminator. If the buffer is too small, it writes as many characters as it can (i.e. nSize), without the null-terminator, and returns the value of the argument nSize.

Description

Copies the system error message into the given buffer with protection around the copy.

2.4.2.11 - MxStringCaseCompare Function

Performs a comparison between two character strings.

Case-insensitive string comparison.

C++

```
int MxStringCaseCompare(IN const char * pszString1, IN const char * pszString2);
```

Parameters

Parameters	Description
IN const char * pszString1	The first null-terminated character string.
IN const char * pszString2	The second null-terminated character string.

Returns

<0, 0 or >0, if the first character string is respectively less than, equal, or greater than the second character string.

Description

Performs a comparison between two character strings, without case sensitivity.

See Also

strcasecmp(3) on Linux manual pages.

2.4.2.12 - MxStringToInt Function

Converts a null-terminated character string into a signed integer, in the given 'base' representation.

C++

```
int64_t MxStringToInt(IN const char* pszBuffer, IN EMxBase eBase = eBASE_DECIMAL, OUT mxt_result* pRes = NULL,
OUT unsigned int* puSizeParsed = NULL);
```

Parameters

Parameters	Description
IN const char* pszBuffer	Input buffer where to read the character string.
IN EMxBase eBase = eBASE_DECIMAL	The base in which pszBuffer is represented. Valid bases are eBASE_BINARY, eBASE_OCTAL, eBASE_DECIMAL, and eBASE_HEXADECIMAL.
OUT mxt_result* pRes = NULL	On exit, contains the result of the conversion.
OUT unsigned int* puSizeParsed = NULL	On exit, contains the number of characters from pszBuffer that have been parsed.

Returns

On success, returns a signed integer and pRes contains resS_OK. On failure, if puSizeParsed is not NULL, returns a signed integer representing the characters that have been parsed. If puSizeParsed is NULL, returns 0. If pRes is not NULL, it contains either resFE_FAIL or resFE_INVALID_ARGUMENT. If an overflow occurs, 0 is returned and pRes is set to resFE_FAIL

Description

Converts a character string into a signed integer in the given 'base' representation.

2.4.2.13 - MxStringToInt Function

Converts a character buffer into a signed integer, in the given 'base' representation.

C++

```
int64_t MxStringToInt(IN const char* pszBuffer, IN unsigned int uSize, IN EMxBase eBase = eBASE_DECIMAL, OUT
mxt_result* pRes = NULL, OUT unsigned int* puSizeParsed = NULL);
```

Parameters

Parameters	Description
IN const char* pszBuffer	Input buffer where to read the character string.
IN unsigned int uSize	The number of characters from pszBuffer to parse.
IN EMxBase eBase = eBASE_DECIMAL	The base in which pszBuffer is represented. Valid bases are eBASE_BINARY, eBASE_OCTAL, eBASE_DECIMAL, and eBASE_HEXADECIMAL.
OUT mxt_result* pRes = NULL	On exit, contains the result of the conversion.
OUT unsigned int* puSizeParsed = NULL	On exit, contains the number of characters from pszBuffer that have been parsed.

Returns

On success, returns a signed integer and pRes contains resS_OK. On failure, if puSizeParsed is not NULL, returns a signed integer representing the characters that have been parsed. If puSizeParsed is NULL, returns 0. If pRes is not NULL, it contains either resFE_FAIL or resFE_INVALID_ARGUMENT. If an overflow occurs, 0 is returned and pRes is set to resFE_FAIL

Description

Converts a character string into a signed integer in the given 'base' representation.

2.4.2.14 - MxStringToUint Function

Converts a null-terminated character string into an unsigned integer, in the given 'base' representation.

C++

```
uint64_t MxStringToUint(IN const char* pszBuffer, IN EMxBase eBase = eBASE_DECIMAL, OUT mxt_result* pRes = NULL,
```

```
OUT unsigned int* puSizeParsed = NULL);
```

Parameters

Parameters	Description
IN const char* pszBuffer	Input string where to read the character string. MUST be '0' terminated.
IN EMxBase eBase = eBASE_DECIMAL	The base in which pcBuffer is represented. Valid bases are eBASE_BINARY, eBASE_OCTAL, eBASE_DECIMAL, and eBASE_HEXADECIMAL.
OUT mxt_result* pRes = NULL	On exit, contains the result of the conversion.
OUT unsigned int* puSizeParsed = NULL	On exit, contains the number of characters from pszBuffer that have been parsed.

Returns

On success, returns an unsigned integer and pRes contains resS_OK. On failure, if puSizeParsed is not NULL, returns an unsigned integer representing the characters that have been parsed. If puSizeParsed is NULL, returns 0. If pRes is not NULL, it contains either resFE_FAIL or resFE_INVALID_ARGUMENT. If an overflow occurs, 0 is returned and pRes is set to resFE_FAIL

Description

Converts a character string into an unsigned integer in the given 'base' representation.

2.4.2.15 - MxStringToInt Function

Converts a character buffer into an unsigned integer, in the given 'base' representation.

C++

```
uint64_t MxStringToInt(IN const char* pszBuffer, IN unsigned int uSize, IN EMxBase eBase = eBASE_DECIMAL, OUT mxt_result* pRes = NULL, OUT unsigned int* puSizeParsed = NULL);
```

Parameters

Parameters	Description
IN const char* pszBuffer	Input buffer where to read the character string.
IN unsigned int uSize	The number of characters from pszBuffer to parse.
IN EMxBase eBase = eBASE_DECIMAL	The base in which pszBuffer is represented. Valid bases are eBASE_BINARY, eBASE_OCTAL, eBASE_DECIMAL, and eBASE_HEXADECIMAL.
OUT mxt_result* pRes = NULL	On exit, contains the result of the conversion.
OUT unsigned int* puSizeParsed = NULL	On exit, contains the number of characters from pszBuffer that have been parsed.

Returns

On success, returns an unsigned integer and pRes contains resS_OK. On failure, if puSizeParsed is not NULL, returns an unsigned integer representing the characters that have been parsed. If puSizeParsed is NULL, returns 0. If pRes is not NULL, it contains either resFE_FAIL or resFE_INVALID_ARGUMENT. If an overflow occurs, 0 is returned and pRes is set to resFE_FAIL

Description

Converts a character string into an unsigned integer in the given 'base' representation.

2.4.2.16 - MxTraceAddOutputHandler Function

Appends the given handler to the list of trace output handlers.

C++

```
bool MxTraceAddOutputHandler(IN mxt_PFNTraceOutputHandler pfnNewHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceOutputHandler pfnNewHandler	A pointer to a function that outputs the traces.

Returns

True if the handler has been successfully added, false otherwise.

Description

Appends the given handler to the list of trace output handlers. Each trace is passed to each registered output handler. This function does not allow duplicates.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.17 - MxTraceDisableAllFields Function

Disables all tracing fields.

C++

```
bool MxTraceDisableAllFields();
```

Returns

Always true.

Description

Disables all tracing fields.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.18 - MxTraceDisableField Function

Disables a tracing field

C++

```
bool MxTraceDisableField(IN EMxTraceField eTraceField);
```

Parameters

Parameters	Description
IN EMxTraceField eTraceField	EMxTraceField (see page 15) containing the field to Disable.

Returns

True if successful, false if it fails.

Description

Disables a specific tracing field.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.19 - MxTraceDisableLevel Function Deprecated since 2.1.6

Removes a tracing level.

C++

```
void MxTraceDisableLevel(IN EMxTraceLevel eLevel);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	The level to remove from the traces

Description

Disables the trace level eLevel for all nodes. The scope of this method is system-wide and affects all nodes. Node specific levels previously set may be affected.

The use of this method is deprecated in favour of the new method MxTraceDisableNodeLevel (see page 25).

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

MxTraceGetLevelsEnabled (see page 29)

2.4.2.20 - MxTraceDisableNode Function

Disables a tracing node.

C++

```
bool MxTraceDisableNode( IN const char* pszTraceNodePath, IN bool bRecursive);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	A path to the tracing node.
IN bool bRecursive	If set to true, disables all child nodes, otherwise disables only the current node.

Returns

True if the node is successfully disabled, otherwise it returns false.

Description

Disables the tracing node matching the passed node path and optionally all its children so the node(s) can be hidden.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceEnableNode](#) (see page 27)

2.4.2.21 - MxTraceDisableNodeLevel Function

Removes a tracing level for this node.

C++

```
bool MxTraceDisableNodeLevel( IN const char* pszTraceNodePath, IN EMxTraceLevel eLevel, IN bool bRecursive);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	The path to the tracing node.
IN EMxTraceLevel eLevel	The trace level to be disabled.
IN bool bRecursive	If set to true, disables the specified trace level for all child nodes, otherwise disables it only for the current node.

Returns

True if the trace level was successfully disabled for the specified node, otherwise it returns false.

Description

Enables the tracing level given by eLevel for the specified node path. Optionally, if bRecursive is set to true, the specified level is disabled for all its children, so the node(s) can be hidden.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceEnableNodeLevel](#) (see page 27)

2.4.2.22 - MxTraceEnableAllFields Function

Enables all tracing fields.

C++

```
bool MxTraceEnableAllFields();
```

Returns

Always true.

Description

Enables all tracing fields.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.23 - MxTraceEnableDefaultFields Function

Enables default tracing fields.

C++

```
bool MxTraceEnableDefaultFields();
```

Returns

Always true.

Description

Enables default tracing fields.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.24 - MxTraceEnableField Function

Enables a tracing field

C++

```
bool MxTraceEnableField(IN EMxTraceField eTraceField);
```

Parameters

Parameters	Description
IN EMxTraceField eTraceField	EMxTraceField (see page 15) containing the field to Enable.

Returns

True if successful, false if it fails.

Description

Enables a specific tracing field.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.25 - MxTraceEnableLevel Function Deprecated since 2.1.6

Adds a tracing level.

C++

```
void MxTraceEnableLevel(IN EMxTraceLevel eLevel);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	The level to add to the traces.

Description

Adds the trace level eLevel for all nodes. The scope of this method is system-wide and affects all nodes. Node specific levels previously set may be affected.

The use of this method is deprecated in favour of the new method MxTraceEnableNodeLevel (see page 27).

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

MxTraceGetLevelsEnabled (see page 29)

2.4.2.26 - MxTraceEnableMaxLevels Function Deprecated since 2.1.6

Adds all tracing levels less than or equal to the level passed.

C++

```
void MxTraceEnableMaxLevels( IN EMxTraceLevel eLevel);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	The maximum level to add to the traces.

Description

Adds all tracing levels that are equal to or lower than eLevel argument for all nodes. The scope of this method is system-wide and affects all nodes. Node specific levels previously set may be affected.

The use of this method is deprecated in favour of the new method MxTraceEnableNodeMaxLevels (see page 28).

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

MxTraceGetLevelsEnabled (see page 29)

2.4.2.27 - MxTraceEnableNode Function

Enables a tracing node.

C++

```
bool MxTraceEnableNode( IN const char* pszTraceNodePath, IN bool bRecursive);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	A path to the tracing node.
IN bool bRecursive	If set to true, enables all child nodes, otherwise enables only the current node.

Returns

True if the node is successfully enabled, otherwise it returns false.

Description

Enables the tracing node matching the passed node path and optionally all its children. When a node, and optionally its children, is enabled all traces made on it will be output.

The node to enable is referred by using the path in the form of the text string pointed by pszTraceNodePath. This is similar to navigating a directory tree in a file system. Moreover, using a path abstracts the node location and gives access to the node without forcing to use the node's variable. This is useful because usually the part of code that controls which traces come out is not the part that uses it. See the example below:

```
// Disable all the nodes in the tree by disabling the root node
// recursively.
MxTraceDisableNode("/", true);

// This trace will not come out.
MX_TRACE2(0, &g_stFramework, "This trace will not be shown...");

// Enable only the Framework node but not its children.
MxTraceEnableNode("/Framework", false);

// Now the trace will come out.
MX_TRACE2(0, &g_stFramework, "This trace will be printed out...");
```

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

General Tracing Configuration (see page 2)

2.4.2.28 - MxTraceEnableNodeLevel Function

Adds a tracing level for this node.

C++

```
bool MxTraceEnableNodeLevel(IN const char* pszTraceNodePath, IN EMxTraceLevel eLevel, IN bool bRecursive);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	The path to the tracing node.
IN EMxTraceLevel eLevel	The trace level to be enabled.
IN bool bRecursive	If set to true, enables the specified trace level for all child nodes, otherwise enables it only for the current node.

Returns

True if the level was successfully enabled, otherwise it returns false.

Description

Enables the trace level defined by eLevel for the specified node path and, optionally, for all its children. When a trace level is enabled for a node, all traces for that level made on it are outputted.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceDisableNodeLevel](#) (see page 25), [MxTraceEnableNodeMaxLevels](#) (see page 28), [struct EMxTraceLevel](#) (see page 16)

2.4.2.29 - MxTraceEnableNodeMaxLevels Function

Adds all tracing levels less than or equal to the level passed for this node.

C++

```
bool MxTraceEnableNodeMaxLevels(IN const char* pszTraceNodePath, IN EMxTraceLevel eLevel, IN bool bRecursive);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	The path to the tracing node.
IN EMxTraceLevel eLevel	The maximum trace level to be enabled.
IN bool bRecursive	If set to true, enables the specified traces levels for all child nodes, otherwise enables them only for the current node.

Returns

True if the trace levels were successfully enabled for the specified node, otherwise it returns false.

Description

Adds, for the specified node path, all the trace levels that are equal to or lower than eLevel. Optionally, if bRecursive is set to true, those levels are also enabled for all the child nodes. When a trace level is enabled for a node, all traces issued on that node and level are outputted.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceEnableNodeLevel](#) (see page 27).

2.4.2.30 - MxTraceEnableSyslogFields Function

Enables syslog tracing fields.

C++

```
bool MxTraceEnableSyslogFields();
```

Returns

Always true.

Description

Enables Syslog tracing fields and preserves previously Enabled tracing fields.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.31 - MxTraceFieldIsEnabled Function

Returns true if a field is active, false if it is not.

C++

```
bool MxTraceFieldIsEnabled(IN EMxTraceField eTraceField);
```

Parameters

Parameters	Description
IN EMxTraceField eTraceField	Contains the field for which the active state verification is made.

Returns

True if the field is active, false otherwise.

Description

Returns true if a field is active, false if it is not.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.32 - MxTraceGetLevelsEnabled Function Deprecated since 2.1.6

Gets the tracing levels currently enabled.

C++

```
EMxTraceLevel MxTraceGetLevelsEnabled();
```

Returns

A bit mask made up of all the levels that are allowed to trace at root level. This is valid at global node scope when only the old level Enable/Disable methods have been used.

Description

Gets the system-wide tracing levels currently allowed for all the nodes. The value returned by this method is only meaningful when strictly used with MxTraceEnableLevel (see page 26), MxTraceDisableLevel (see page 24), and MxTraceEnableMaxLevels (see page 26).

Notes: 1)The use of the system-wide methods MxTraceEnableLevel (see page 26), MxTraceDisableLevel (see page 24), MxTraceEnableMaxLevels (see page 26), and MxTraceGetLevelsEnabled is deprecated in favour of the node specific methods:

- MxTraceEnableNodeLevel (see page 27)
- MxTraceDisableNodeLevel (see page 25)
- MxTraceEnableNodeMaxLevels (see page 28)
- MxTraceGetNodeLevelsEnabled (see page 29)

2)It is discouraged to mix use of the system-wide and the node specific methods for controlling the tracing levels as it may result in incoherent behaviour. System-wide methods are left only for backward compatibility.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

MxTraceEnableLevel (see page 26), MxTraceDisableLevel (see page 24), MxTraceEnableMaxLevels (see page 26)

2.4.2.33 - MxTraceGetNodeLevelsEnabled Function

Gets the tracing levels currently enabled for this node.

C++

```
EMxTraceLevel MxTraceGetNodeLevelsEnabled(IN const char* pszTraceNodePath);
```

Parameters

Parameters	Description
IN const char* pszTraceNodePath	The path to the tracing node.

Returns

A bit mask made up of all the levels that are allowed to trace for the specified node.

Description

Gets the tracing levels currently allowed to trace for the specified node path. Note that it may return tracing levels that were excluded from the compilation, in which case they should be ignored.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceEnableNodeLevel](#) (see page 27)

2.4.2.34 - MxTracePrintTree Function

Prints all enabled trace nodes of the tree.

C++

```
void MxTracePrintTree();
```

Description

Prints all the trace nodes of the tree and their state.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.35 - MxTraceRegisterNode Function

Registers a tracing node.

C++

```
bool MxTraceRegisterNode(IN STraceNode* pstparentNode, IN STraceNode* pstNode, IN const char* psznodeName);
```

Parameters

Parameters	Description
IN STraceNode* pstparentNode	A pointer to the parent trace node.
IN STraceNode* pstNode	A pointer to the trace node to register.
IN const char* psznodeName	A pointer to a null-terminated string containing the name of the node.

Returns

True if the node is successfully registered, otherwise it returns false.

Description

Registers the node pointed by pstNode as one child of the node pointed by pstparentNode inside the whole tracing tree. The null-terminated string pointed by psznodeName is held inside the node. The node name represents a path part of a "path" used to reference the node when it is required to enable or disable the tracing node.

Register the first node of a branch to the root node g_stTraceRoot (see page 7), which is always available. Then continue the registration of related nodes as children in the new branch created as in the following example:

```
// Declare the nodes.
STraceNode stNewProduct;
STraceNode stNewProductFeature1;
STraceNode stNewProductFeature1CSender;
STraceNode stNewProductFeature2;

// Initialization process
...
MxTraceRegisterNode(&g_stTraceRoot, &stNewProduct, "NewProduct");
MxTraceRegisterNode(&stNewProduct, &stNewProductFeature1, "Feature1");
MxTraceRegisterNode(&stNewProductFeature1, &stNewProductFeature1CSender, "CSender");
MxTraceRegisterNode(&stNewProduct, &stNewProductFeature2, "Feature2");
```

```

// Feature code....
MX_TRACE2(0, &stNewProductFeature1, "Trace this and that...");

// Uninitialization process
MxTraceUnregisterNode(&g_stTraceRoot, &stNewProduct);
...

```

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

General Tracing Configuration (See page 2)

2.4.2.36 - MxTraceRemoveOutputHandler Function

Removes each occurrence of the given function pointer in the list of registered trace output handlers.

C++

```
void MxTraceRemoveOutputHandler(IN mxt_PFNTraceOutputHandler pfnHandlerToRemove);
```

Parameters

Parameters	Description
IN mxt_PFNTraceOutputHandler pfnHandlerToRemove	A pointer to the function that should be removed from the list.

Description

Removes each occurrence of the given function pointer in the list of registered trace output handlers.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.37 - MxTraceSetHostNameField Function

Sets the local hostname field.

C++

```
bool MxTraceSetHostNameField(IN const char* pszDestinationAddress);
```

Parameters

Parameters	Description
IN const char* pszDestinationAddress	IP address of the destination address where traces are sent.

Returns

true

Description

Sets the traces hostname field as the local IP address that is used to communicate with the destination where traces are sent, if they are sent through the network.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.38 - MxTraceSetNewCallStackHandler Function

Configures a new call stack trace handler.

C++

```
mxt_PFNTraceCallStackHandler MxTraceSetNewCallStackHandler(IN mxt_PFNTraceCallStackHandler pfnNewHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceCallStackHandler pfnNewHandler	A pointer to a function that is in charge of building and outputting the call stack trace. This parameter can safely be NULL and is internally registered to a dummy function.

Returns

A pointer to the function that was previously registered or NULL.

Description

Replaces the current call stack handler (the function that produces the call stack trace) by the new one given.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.39 - MxTraceSetNewFormatHandler Function

Configures a new format handler.

C++

```
mxt_PFNTraceFormatHandler MxTraceSetNewFormatHandler(IN mxt_PFNTraceFormatHandler pfnHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceFormatHandler pfnHandler	A pointer to a function that is in charge of building and outputting the whole trace. This parameter can safely be NULL and is internally, registered to a dummy function.

Returns

A pointer to the function that was previously registered or NULL.

Description

Replaces the current format handler (the function that builds the trace) by the new one given.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.40 - MxTraceSetNewOutputHandler Function

Sets a new output handler to replace the previous ones.

C++

```
void MxTraceSetNewOutputHandler(IN mxt_PFNTraceOutputHandler pfnNewHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceOutputHandler pfnNewHandler	A pointer to a function that outputs the traces.

Description

Replaces all the registered output handlers by the given one.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.2.41 - MxTraceSetSysTimeFormatHandler Function

Configures a new system time format handler.

C++

```
mxt_PFNTraceGenericFormatHandler MxTraceSetSysTimeFormatHandler(IN mxt_PFNTraceGenericFormatHandler pfnNewHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceGenericFormatHandler pfnNewHandler	A pointer to a function that is in charge of getting and formatting the timestamp. This parameter can safely be NULL.

Returns

A pointer to the function that was previously registered or NULL.

Description

Replaces the current system up time format handler (the function that builds the system up timestamp in the trace) by the new one given. When set to NULL, disallows the system up time field (<21>) to appear in the traces.

Location

Defined in Basic/MxTrace.cpp.

2.4.2.42 - MxTraceSetTimeFormatHandler Function

Configures a new time stamp format handler.

C++

```
mxt_PFNTraceGenericFormatHandler MxTraceSetTimeFormatHandler( IN mxt_PFNTraceGenericFormatHandler pfnNewHandler);
```

Parameters

Parameters	Description
IN mxt_PFNTraceGenericFormatHandler pfnNewHandler	A pointer to a function that is in charge of getting and formatting the timestamp. This parameter can safely be NULL.

Returns

A pointer to the function that was previously registered or NULL.

Description

Replaces the current time format handler (the function that builds the timestamp in the trace) by the new one given. When set to NULL, disallows the local date and time field (<02>) to appear in the traces.

Location

Defined in Basic/MxTrace.cpp.

2.4.2.43 - MxTraceToDebugger Function

mxt_PFNTraceOutputHandler (see page 92) for writing traces to a debugger.

C++

```
void MxTraceToDebugger( IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN const char* pszTrace, IN int nMsgSize);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN uint32_t uTraceUniqueId	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN const char* pszTrace	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN int nMsgSize	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.

Description

This trace output handler outputs the trace to the debugger. This currently works only on MSVC. Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for parameters description.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

MxTraceToStdout (see page 34), MxTraceToDebugger

2.4.2.44 - MxTraceToStderr Function

mxt_PFNTraceOutputHandler (see page 92) for writing traces to the standard error.

C++

```
void MxTraceToStderr( IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN const char* pszTrace, IN int nMsgSize);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN uint32_t uTraceUniqueId	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN const char* pszTrace	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN int nMsgSize	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.

Description

This trace output handler outputs the trace to the standard error. Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for parameters description.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceToStdout](#) (see page 34), [MxTraceToDebugger](#) (see page 33)

2.4.2.45 - MxTraceToStdout Function

`mxt_PFNTraceOutputHandler` (see page 92) for writing traces to the standard output.

C++

```
void MxTraceToStdout(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN const char* pszTrace, IN int nMsgSize);
```

Parameters

Parameters	Description
IN EMxTraceLevel eLevel	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN uint32_t uTraceUniqueId	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN const char* pszTrace	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.
IN int nMsgSize	Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for the parameter description.

Description

This trace output handler outputs the trace to the standard output. Refer to the description of mxt_PFNTraceOutputHandler (see page 92) for parameters description.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceToStderr](#) (see page 33), [MxTraceToDebugger](#) (see page 33)

2.4.2.46 - MxTraceUnregisterNode Function

Unregisters a tracing node.

C++

```
bool MxTraceUnregisterNode(IN STraceNode* pstparentNode, IN STraceNode* pstNode);
```

Parameters

Parameters	Description
IN STraceNode* pstparentNode	A pointer to the parent trace node.
IN STraceNode* pstNode	A pointer to the trace node to unregister.

Returns

True if the node is successfully unregistered, otherwise it returns false.

Description

Unregisters the node pointed by `pstNode` that was one child of the node pointed by `pstparentNode` inside the whole tracing tree.

Usually, only the first node of a branch registered as a child of the root node `g_stTraceRoot` (see page 7) is unregistered. If the node has children, they are automatically unregistered.

Location

Defined in `Basic/MxTrace.h` but must include `Config/MxConfig.h` to access this.

See Also

`MxTraceRegisterNode` (see page 30)

2.4.2.47 - `MxUintToString` Function

Converts an unsigned integer into a character string, in the given 'base' representation.

C++

```
char* MxUintToString(IN uint64_t uValue, IN unsigned int uCapacity, OUT char* pszBuffer, OUT unsigned int* puSize, IN EMxBase eBase = eBASE_DECIMAL);
```

Parameters

Parameters	Description
IN uint64_t uValue	The unsigned integer to convert.
IN unsigned int uCapacity	Maximum number of characters that can be written into the output buffer (including the null-terminator).
OUT char* pszBuffer	Output buffer where to write the formatted character string.
OUT unsigned int* puSize	On exit, contains the number of characters written to <code>pszBuffer</code> , excluding the null-terminator.
IN EMxBase eBase = eBASE_DECIMAL	The base in which <code>uValue</code> is represented. Valid bases are <code>eBASE_BINARY</code> , <code>eBASE_OCTAL</code> , <code>eBASE_DECIMAL</code> , and <code>eBASE_HEXADECIMAL</code> .

Returns

On success, returns `pszBuffer` and `puSize` contains the number of characters written to `pszBuffer`, excluding the null-terminator. On failure, returns `NULL`.

Description

Converts an unsigned integer into a character string in the given 'base' representation.

2.4.2.48 - `MxVsnprintf` Function

Writes into a buffer a formatted character string.

C++

```
int MxVsnprintf(OUT char * pszBuffer, IN int nSize, IN const char * pszFormat, INOUT va_list args);
```

Parameters

Parameters	Description
OUT char * pszBuffer	Output buffer where to write the formatted character string.
IN int nSize	Maximum number of characters that can be written in the output buffer (including the null-terminator).
IN const char * pszFormat	A character string that specifies how subsequent arguments (accessed by 'args') are converted to the output buffer.
INOUT va_list args	The arguments in a 'va_list'. This function does not call <code>va_end</code> , so the state of 'args' is unknown after it returns.

Returns

Returns the number of characters written to the buffer, excluding the null-terminator. If the buffer is too small, it writes as many characters as it can (i.e., `nSize`), without a null-terminator, and returns the value of the argument `nSize`.

Description

Writes a formatted character string into a buffer. This function is portable across all supported OSs and behaves identically except for the format (`pszFormat`), which is handled differently by each OS.

See Also

Linux manual pages of `vsnprintf(3)`, which differ only by the return value.

2.4.2.49 - NotifyEventObservers Function

Notifies observers.

C++

```
void NotifyEventObservers(IN EEventNotifier eEvent, ...);
```

Parameters

Parameters	Description
IN EEventNotifier eEvent	The ID of the event of which the observer must be notified. Variable parameters (dependent on the event). Refer to the enum definition for more information about the required parameters.

Description

Called by someone who wants to notify that an event happened.

2.4.2.50 - RegisterEventObserver Function

Registers an observer.

C++

```
void RegisterEventObserver(IN mxt_pfnEventObserver pfnEventObserver, IN void* pContext = NULL);
```

Parameters

Parameters	Description
IN mxt_pfnEventObserver pfnEventObserver	The observer to register.
IN void* pContext = NULL	The context associated with the observer. The user needs to call UnregisterEventObserver (see page 36) with the same context.

Description

Registers an observer. Does nothing if the observer is already registered.

2.4.2.51 - UnInstallGenericFaultHandler Function

Uninstalls fault handler.

C++

```
bool UnInstallGenericFaultHandler();
```

Description

Under certain OSs and for a limited number of architectures, un-installs fault handlers. Called by Finalize if MXD_FAULT_HANDLER_ENABLE is defined. Otherwise, fault handlers are not installed if InstallGenericFaultHandler (see page 19) is not called.

Location

Defined in Basic/MxFault.h.

2.4.2.52 - UnregisterEventObserver Function

Unregisters an observer.

C++

```
void UnregisterEventObserver(IN mxt_pfnEventObserver pfnEventObserver, IN void* pContext = NULL);
```

Parameters

Parameters	Description
IN mxt_pfnEventObserver pfnEventObserver	The observer to unregister.
IN void* pContext = NULL	The context associated with the observer. The context was previously passed the RegisterEventObserver (see page 36).

Description

Unregisters an observer. Does nothing if the observer is not registered.

2.4.3 - Macros

This section documents the macros of the Sources/Basic folder.

Macros

Macro	Description
GO (see page 39)	Defines parameter direction and ownership.
IN (see page 39)	Defines parameter direction and ownership.
INOUT (see page 39)	Defines parameter direction and ownership.
MX_ASSERT (see page 39)	Basic assertion macro.
MX_ASSERT_EX (see page 40)	Extended basic assertion macro.
MX_ASSERT_ONLY (see page 40)	Used to declare variables used in assert macros.
MX_ASSERT_ONLY_RT (see page 40)	Used to declare variables used in real time assert macros.
MX_ASSERT_PERROR (see page 41)	Macro asserting on errno.
MX_ASSERT_PERROR_EX (see page 41)	Extended macro asserting on errno.
MX_ASSERT_PERROR_RT (see page 41)	Real-time macro asserting on errno.
MX_ASSERT_PERROR_RT_EX (see page 42)	Extended real-time macro asserting on errno.
MX_ASSERT_RT (see page 42)	Real-time assertion macro.
MX_ASSERT_RT_EX (see page 42)	Extended real-time assertion macro.
MX_DEFINEINT64 (see page 43)	Defines a 64 bits integer by adding the required suffix.
MX_DEFINEUINT64 (see page 43)	Defines a 64 bits unsigned integer by adding the required suffix.
MX_HI16 (see page 43)	Gets the hi 16 bits portion of an integer.
MX_HI8 (see page 44)	Gets the hi 8 bits portion of an integer.
MX_HTON64 (see page 44)	Macro to convert a 64 bit word from host to network byte order.
MX_HTONL (see page 44)	Macro to convert a 32 bit unsigned integer from host to network byte order.
MX_HTONS (see page 44)	Macro to convert a 16 bit unsigned integer from host to network byte order.
MX_LOW16 (see page 45)	Gets the low 16 bits portion of an integer.
MX_LOW32 (see page 45)	Gets the low 32 bits portion of an integer.
MX_LOW8 (see page 45)	Gets the low 8 bits portion of an integer.
MX_MAKEUINT16 (see page 46)	Makes a 16 bits unsigned integer from two 8 bits, hi and low portions.
MX_MAKEUINT32 (see page 46)	Makes a 32 bits unsigned integer from two 16 bits, hi and low portions.
MX_MAX (see page 46)	Returns the largest of two values.
MX_MIN (see page 46)	Returns the smallest of two values.
MX_NAMESPACE (see page 47)	Specifies a namespace for a variable or method.
MX_NAMESPACE_END (see page 47)	Ends a namespace.
MX_NAMESPACE_START (see page 47)	Starts a namespace.
MX_NAMESPACE_USE (see page 48)	Uses a namespace.
MX_NTOH64 (see page 48)	Macro to convert a 64 bit word from network to host byte order.
MX_NTOHL (see page 48)	Macro to convert a 32 bit unsigned integer from network to host byte order.
MX_NTOHS (see page 48)	Macro to convert a 16 bit unsigned integer from network to host byte order.
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_BEGIN (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_END (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_FAIL_ERROR_MSG_TBL_BEGIN (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_FAIL_ERROR_MSG_TBL_END (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_SUCCESS_INFO_MSG_TBL_END (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_BEGIN (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_END (see page 49)	Macros used to automatically generate messages table initialization for a package.
MX_RGET_CID (see page 50)	Result manipulation macro.
MX_RGET_LEV (see page 50)	Result manipulation macro.
MX_RGET_MSG_STR (see page 50)	Result manipulation macro.
MX_RGET_PID (see page 50)	Result manipulation macro.
MX_RGET_PKG_BASE_FAIL_CRITICAL_CODE_ID (see page 49)	Macro used to get the lower value of the result code ID enumeration for a unique package ID.
MX_RGET_PKG_BASE_FAIL_ERROR_CODE_ID (see page 49)	Macro used to get the lower value of the result code ID enumeration for a unique package ID.
MX_RGET_PKG_BASE_SUCCESS_WARNING_CODE_ID (see page 49)	Macro used to get the lower value of the result code ID enumeration for a unique package ID.
MX_RGET_WORST_OF (see page 50)	Result manipulation macro.
MX_RIS_F (see page 50)	Result manipulation macro.

MX_RIS_FC (see page 50)	Result manipulation macro.
MX_RIS_FE (see page 50)	Result manipulation macro.
MX_RIS_S (see page 50)	Result manipulation macro.
MX_RIS_SI (see page 50)	Result manipulation macro.
MX_RIS_SW (see page 50)	Result manipulation macro.
MX_SIZEOFAARRAY (see page 51)	Returns the size of an array.
MX_SWAPBYTES16 (see page 51)	Swaps the hi and low bytes within a 16 bits unsigned integer.
MX_SWAPBYTES32 (see page 52)	Swaps the hi and low bytes within a 32 bits unsigned integer.
MX_TRACE (see page 52)	Standard tracing macros.
MX_TRACE_ALL_CALLSTACKS (see page 53)	Call stack tracing macros.
MX_TRACE_CALLSTACK (see page 53)	Call stack tracing macros.
MX_TRACE_CALLSTACK_COREDUMP (see page 53)	Call stack tracing macros.
MX_TRACE_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE0 (see page 52)	Standard tracing macros.
MX_TRACE0_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE0_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE1 (see page 52)	Standard tracing macros.
MX_TRACE1_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE1_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE2 (see page 52)	Standard tracing macros.
MX_TRACE2_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE2_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE3 (see page 52)	Standard tracing macros.
MX_TRACE3_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE3_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE4 (see page 52)	Standard tracing macros.
MX_TRACE4_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE4_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE5 (see page 52)	Standard tracing macros.
MX_TRACE5_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE5_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE6 (see page 52)	Standard tracing macros.
MX_TRACE6_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE6_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE7 (see page 52)	Standard tracing macros.
MX_TRACE7_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE7_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE8 (see page 52)	Standard tracing macros.
MX_TRACE8_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE8_IS_ENABLED (see page 54)	Tracing level activation check macros.
MX_TRACE9 (see page 52)	Standard tracing macros.
MX_TRACE9_HEX (see page 54)	Hexadecimal tracing macros.
MX_TRACE9_IS_ENABLED (see page 54)	Tracing level activation check macros.
OUT (see page 39)	Defines parameter direction and ownership.
TOA (see page 39)	Defines parameter direction and ownership.
TOS (see page 39)	Defines parameter direction and ownership.
MX_MAKEUINT64 (see page 57)	Makes a 64 bits unsigned integer from two 32 bits, hi and low portions.
MX_HI32 (see page 57)	Gets the hi 32 bits portion of an integer.
MX_SWAPBYTES64 (see page 57)	Swaps the hi and low bytes within a 64 bits unsigned integer.
MX_ERRNO_SET (see page 57)	Sets the errno value.
MX_ERRNO_GET (see page 58)	Gets the errno value.
MX_REMOVE_UNUSED_FUNCTION_PARAM_WARNING (see page 58)	Removes warning about unused function parameters.
MX_MAKE_STRING_NULL_SAFE (see page 58)	Converts a NULL string to its system-dependent equivalent.
MX_MAKE_NULL_EMPTY_STRING (see page 58)	Converts a NULL string to an empty string.
MX_ALIGNMENT_OF (see page 59)	Returns a value, of type size_t, that is the alignment requirement of the given type.
MX_VOIDPTR_TO_OPQ (see page 59)	Converts a void pointer to an opaque (mxt_opaque (see page 87)).
MX_OPQ_TO_VOIDPTR (see page 59)	Converts an opaque (mxt_opaque (see page 87)) to a void pointer.
MX_INT32_TO_OPQ (see page 60)	Converts a 32 bits signed integer to an opaque (mxt_opaque (see page 87)).
MX_OPQ_TO_INT32 (see page 60)	Converts an opaque (mxt_opaque (see page 87)) to a 32 bits signed integer.

MX_UINT32_TO_OPQ (see page 61)	Converts a 32 bits unsigned integer to an opaque (mxt_opaque (see page 87)).
MX_OPQ_TO_UINT32 (see page 61)	Converts an opaque (mxt_opaque (see page 87)) to a 32 bits unsigned integer.
MX_ISDIGIT (see page 62)	Verifies if a character is alphabetic or numeric.

2.4.3.1 - Parameters qualifier

Defines parameter direction and ownership.

C++

```
#define IN
#define OUT
#define INOUT
#define TOA
#define TOS
#define GO
```

Description

Modifiers help to eliminate confusion around parameter direction and ownership acquisition. The modifiers for direction and ownership can be combined. Modifiers are used in the member function signature but may also be used in function call to eliminate ambiguity.

IN:

An input parameter; not modified inside the caller scope.

OUT:

An output parameter; modified to communicate information to the caller.

INOUT:

An input and output parameter.

TOA:

Indicates that the ownership of the pointer is always taken from the caller. The caller is no longer responsible for the object deletion.

TOS:

Indicates that the ownership of the pointer is only taken from the caller on success. The caller is no longer responsible for the object deletion.

GO:

Indicates that the ownership of the pointer is given to the caller. The caller becomes responsible for the object deletion.

See Also

Deprecated parameters qualifier (see page 62)

2.4.3.2 - MX_ASSERT Macro

Basic assertion macro.

C++

```
#define MX_ASSERT(assertion)
```

Parameters

Parameters	Description
assertion	Expression to assert.

Description

Basic assertion macro. The macro outputs data only if the expression passed is false.

```
MX_ASSERT(i < m_vecItems.Size());
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.3 - MX_ASSERT Macro

Extended basic assertion macro.

C++

```
#define MX_ASSERT(assertion, string)
```

Parameters

Parameters	Description
assertion	Expression to assert.
string	String to output with the assertion.

Description

Extended basic assertion macro. Other than the assertion, takes an extra comment to clarify the assertion. If the assertion fails, the comment is included in the assertion trace. The macro outputs data only if the expression passed is false.

```
MX_ASSERT_EX(i < m_vecItems.Size(), "Illegal access on m_vecItems.");
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.4 - MX_ASSERT_ONLY Macro

Used to declare variables used in assert macros.

C++

```
#define MX_ASSERT_ONLY
```

Parameters

Parameters	Description
assertion	Variable to declare.

Description

Used to declare variables used in assertion macros only. This is to avoid annoying compiler warnings for unused variables.

```
MX_ASSERT_ONLY(bool bValid = false);
MX_ASSERT(bValid);
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.5 - MX_ASSERT_ONLY_RT Macro

Used to declare variables used in real time assert macros.

C++

```
#define MX_ASSERT_ONLY_RT
```

Parameters

Parameters	Description
assertion	Variable to declare.

Description

Used to declare variables used in real time assertion macros only. This is to avoid annoying compiler warnings for unused variables.

```
MX_ASSERT_ONLY_RT(bool bValid = false);
MX_ASSERT_RT(bValid);
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.6 - MX_ASSERT_PERROR Macro

Macro asserting on errno.

C++

```
#define MX_ASSERT_PERROR
```

Description

Macro asserting on errno.

NOTES: All the assertion macros involving `errno` described above should be used with care since not all functions set it. Moreover, some functions set `errno` only on a specific operating system. If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.7 - MX_ASSERT_PERROR_EX Macro

Extended macro asserting on errno.

C++

```
#define MX_ASSERT_PERROR_EX(string)
```

Parameters

Parameters	Description
string	String to print out.

Description

Extended macro asserting on errno.

```
MX_ASSERT_PERROR_EX("send() failed!");
```

NOTES: All the assertion macros involving `errno` described above should be used with care since not all functions set it. Moreover, some functions set `errno` only on a specific operating system. If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.8 - MX_ASSERT_PERROR_RT Macro

Real-time macro asserting on errno.

C++

```
#define MX_ASSERT_PERROR_RT
```

Description

Real-time macro asserting on errno.

NOTES: All the assertion macros involving `errno` described above should be used with care since not all functions set it. Moreover, some functions set `errno` only on a specific operating system. If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.9 - MX_ASSERT_PERROR_RT_EX Macro

Extended real-time macro asserting on errno.

C++

```
#define MX_ASSERT_PERROR_RT_EX(string)
```

Parameters

Parameters	Description
string	String to print out.

Description

Extended real-time macro asserting on errno.

```
MX_ASSERT_PERROR_RT_EX("send() failed!");
```

Notes

All the assertion macros involving `errno` described above should be used with care since not all functions set it. Moreover, some functions set `errno` only on a specific operating system. If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.10 - MX_ASSERT_RT Macro

Real-time assertion macro.

C++

```
#define MX_ASSERT_RT(assertion)
```

Parameters

Parameters	Description
assertion	Expression to assert.

Description

Real-time assertion macro. The macro outputs data only if the expression passed is false. This assertion is used in time critical code so it can be added or removed easily to avoid adding latency to the execution.

```
MX_ASSERT_RT(m_rtpPacket.IsValid());
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.11 - MX_ASSERT_RT_EX Macro

Extended real-time assertion macro.

C++

```
#define MX_ASSERT_RT_EX(assertion, string)
```

Parameters

Parameters	Description
assertion	Expression to assert.
string	String to output with the assertion.

Description

Extended real-time assertion macro. Other than the assertion, takes an extra comment to clarify the assertion. This assertion is used in time critical code so it can be added or removed easily to avoid adding latency to the execution. If the assertion fails, the comment is included in the assertion trace. The macro outputs data only if the expression passed is false.

```
MX_ASSERT_EX(i < m_vecItems.Size(), "Illegal access on m_vecItems.");
```

NOTES: If an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

Location

Basic/MxAssert.h

2.4.3.12 - MX_DEFINEINT64 Macro

Defines a 64 bits integer by adding the required suffix.

C++

```
#define MX_DEFINEINT64(u64bitVal)
```

Parameters

Parameters	Description
u64bitVal	64 bits value to define.

Description

Defines a 64 bits integer by adding the required suffix. For example, instead of using:

```
const int64_t u64var = 0x123456789abcdef0i64; // with MSVC
const int64_t u64var = 0x123456789abcdef0LL; // with GCC
```

use:

```
const int64_t u64var = MX_DEFINEINT64(0x123456789abcdef0);
```

See Also

[MX_DEFINEUINT64](#) (see page 43)

2.4.3.13 - MX_DEFINEUINT64 Macro

Defines a 64 bits unsigned integer by adding the required suffix.

C++

```
#define MX_DEFINEUINT64(u64bitVal)
```

Parameters

Parameters	Description
u64bitVal	64 bits value to define.

Description

Defines a 64 bits unsigned integer by adding the required suffix. For example, instead of using:

```
const uint64_t u64var = 0x123456789abcdef0ui64; // with MSVC
const uint64_t u64var = 0x123456789abcdef0ULL; // with GCC
```

use:

```
const uint64_t u64var = MX_DEFINEUINT64(0x123456789abcdef0);
```

See Also

[MX_DEFINEINT64](#) (see page 43)

2.4.3.14 - MX_HI16 Macro

Gets the hi 16 bits portion of an integer.

C++

```
#define MX_HI16(n) (static_cast<uint16_t>((static_cast<uint32_t>(n) >> 16) & 0x0000FFFF))
```

Parameters

Parameters	Description
n	Integer from which to get the highest 16 bits.

Description

Gets the hi 16 bits portion of an integer.

See Also

[MX_LOW8](#) (see page 45), [MX_LOW16](#) (see page 45), [MX_LOW32](#) (see page 45), [MX_HI8](#) (see page 44), [MX_HI32](#) (see page

57)

2.4.3.15 - MX_HI8 Macro

Gets the hi 8 bits portion of an integer.

C++

```
#define MX_HI8(n) (static_cast<uint8_t>((static_cast<uint16_t>(n) >> 8) & 0x00FF))
```

Parameters

Parameters	Description
n	Integer from which to get the highest 8 bits.

Description

Gets the hi 8 bits portion of an integer.

See Also

MX_LOW8 (see page 45), MX_LOW16 (see page 45), MX_LOW32 (see page 45), MX_HI16 (see page 43), MX_HI32 (see page 57)

2.4.3.16 - MX_HTON64 Macro

Macro to convert a 64 bit word from host to network byte order.

C++

```
#define MX_HTON64(uParam) (uint64_t)hton64(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to network byte order.

Returns

The value in network byte order.

Description

Macro used to convert a 64 bit word from host to network byte order. It is guaranteed to always return a value of uint64_t (see page 85) type.

2.4.3.17 - MX_HTONL Macro

Macro to convert a 32 bit unsigned integer from host to network byte order.

C++

```
#define MX_HTONL(uParam) (uint32_t)htonl(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to network byte order.

Returns

The value in network byte order.

Description

Macro used to convert a 32 bit unsigned integer from host to network byte order. It is guaranteed to always return a value of uint32_t (see page 85) type.

2.4.3.18 - MX_HTONS Macro

Macro to convert a 16 bit unsigned integer from host to network byte order.

C++

```
#define MX_HTONS(uParam) (uint16_t)htons(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to network byte order.

Returns

The value in network byte order.

Description

Macro used to convert a 16 bit unsigned integer from host to network byte order. It is guaranteed to always return a value of `uint16_t` (see page 85) type.

2.4.3.19 - MX_LOW16 Macro

Gets the low 16 bits portion of an integer.

C++

```
#define MX_LOW16(n) (static_cast<uint16_t>(n))
```

Parameters

Parameters	Description
n	Integer from which to get the lowest 16 bits.

Description

Gets the low 16 bits portion of an integer.

See Also

`MX_LOW8` (see page 45), `MX_LOW32` (see page 45), `MX_HI8` (see page 44), `MX_HI16` (see page 43), `MX_HI32` (see page 57)

2.4.3.20 - MX_LOW32 Macro

Gets the low 32 bits portion of an integer.

C++

```
#define MX_LOW32(n) (static_cast<uint32_t>(n))
```

Parameters

Parameters	Description
n	Integer from which to get the lowest 32 bits.

Description

Gets the low 32 bits portion of an integer.

See Also

`MX_LOW8` (see page 45), `MX_LOW16` (see page 45), `MX_HI8` (see page 44), `MX_HI16` (see page 43), `MX_HI32` (see page 57)

2.4.3.21 - MX_LOW8 Macro

Gets the low 8 bits portion of an integer.

C++

```
#define MX_LOW8(n) (static_cast<uint8_t>(n))
```

Parameters

Parameters	Description
n	Integer from which to get the lowest 8 bits.

Description

Gets the low 8 bits portion of an integer.

See Also

`MX_LOW16` (see page 45), `MX_LOW32` (see page 45), `MX_HI8` (see page 44), `MX_HI16` (see page 43), `MX_HI32` (see page 57)

2.4.3.22 - MX_MAKEUINT16 Macro

Makes a 16 bits unsigned integer from two 8 bits, hi and low portions.

C++

```
#define MX_MAKEUINT16(high, low) static_cast<uint16_t>( static_cast<uint16_t>(high) << 8 | static_cast<uint8_t>(low) )
```

Parameters

Parameters	Description
high	Higher 8 bits of the unsigned integer to define.
low	Lower 8 bits of the unsigned integer to define.

Description

Makes a 16 bits unsigned integer from two 8 bits, hi and low portions.

See Also

[MX_MAKEUINT32](#) (see page 46), [MX_MAKEUINT64](#) (see page 57)

2.4.3.23 - MX_MAKEUINT32 Macro

Makes a 32 bits unsigned integer from two 16 bits, hi and low portions.

C++

```
#define MX_MAKEUINT32(high, low) static_cast<uint32_t>( static_cast<uint32_t>(high) << 16 | static_cast<uint16_t>(low) )
```

Parameters

Parameters	Description
high	Higher 16 bits of the unsigned integer to define.
low	Lower 16 bits of the unsigned integer to define.

Description

Makes a 32 bits unsigned integer from two 16 bits, hi and low portions.

See Also

[MX_MAKEUINT16](#) (see page 46), [MX_MAKEUINT64](#) (see page 57)

2.4.3.24 - MX_MAX Macro

Returns the largest of two values.

C++

```
#define MX_MAX(x,y) ((x) > (y) ? (x) : (y))
```

Parameters

Parameters	Description
x	First value to compare.
y	Second value to compare.

Description

Returns the largest of two values.

See Also

[MX_MIN](#) (see page 46).

2.4.3.25 - MX_MIN Macro

Returns the smallest of two values.

C++

```
#define MX_MIN(x,y) ((x) > (y) ? (y) : (x))
```

Parameters

Parameters	Description
x	First value to compare.
y	Second value to compare.

Description

Returns the smallest of two values.

See Also

[MX_MAX](#) (see page 46).

2.4.3.26 - MX_NAMESPACE Macro

Specifies a namespace for a variable or method.

C++

```
#define MX_NAMESPACE(x) x::
```

Parameters

Parameters	Description
x	Name of the namespace to use.

Description

When namespacing is enabled, specifies a namespace for a variable or a method. For instance, if nTest is a variable defined within the MxTest namespace, it can be referenced by MX_NAMESPACE(MxTest)nTest from outside the MxTest namespace.

See Also

[MXD_ENABLE_NAMESPACE](#) (see page 317), [MX_NAMESPACE_START](#) (see page 47), [MX_NAMESPACE_END](#) (see page 47)

2.4.3.27 - MX_NAMESPACE_END Macro

Ends a namespace.

C++

```
#define MX_NAMESPACE_END(x) }
```

Parameters

Parameters	Description
x	Name of the namespace to end.

Description

When namespacing is enabled, marks the end of a code section to be interpreted within a certain namespace.

See Also

[MXD_ENABLE_NAMESPACE](#) (see page 317), [MX_NAMESPACE_START](#) (see page 47), [MX_NAMESPACE_USE](#) (see page 48)

2.4.3.28 - MX_NAMESPACE_START Macro

Starts a namespace.

C++

```
#define MX_NAMESPACE_START(x) namespace x {
```

Parameters

Parameters	Description
x	Name of the namespace to start.

Description

When namespacing is enabled, marks the beginning of a code section to be interpreted within a certain namespace.

See Also

[MXD_ENABLE_NAMESPACE](#) (see page 317), [MX_NAMESPACE_END](#) (see page 47), [MX_NAMESPACE_USE](#) (see page 48)

2.4.3.29 - MX_NAMESPACE_USE Macro

Uses a namespace.

C++

```
#define MX_NAMESPACE_USE(x) using namespace x;
```

Parameters

Parameters	Description
x	Name of the namespace to use.

Description

When namespacing is enabled, notifies the compiler that the following code section can use variables or methods that are defined in another namespace.

See Also

MXD_ENABLE_NAMESPACE (see page 317), MX_NAMESPACE_START (see page 47), MX_NAMESPACE_END (see page 47)

2.4.3.30 - MX_NTOH64 Macro

Macro to convert a 64 bit word from network to host byte order.

C++

```
#define MX_NTOH64(uParam) (uint64_t)ntoh64(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to host byte order.

Returns

The value in host byte order.

Description

Macro used to convert a 64 bit word from network to host byte order. It is guaranteed to always return a value of uint64_t (see page 85) type.

2.4.3.31 - MX_NTOHL Macro

Macro to convert a 32 bit unsigned integer from network to host byte order.

C++

```
#define MX_NTOHL(uParam) (uint32_t)ntohl(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to host byte order.

Returns

The value in host byte order.

Description

Macro used to convert a 32 bit unsigned integer from network to host byte order. It is guaranteed to always return a value of uint32_t (see page 85) type.

2.4.3.32 - MX_NTOHS Macro

Macro to convert a 16 bit unsigned integer from network to host byte order.

C++

```
#define MX_NTOHS(uParam) (uint16_t)ntohs(uParam)
```

Parameters

Parameters	Description
uParam	Value to convert to host byte order.

Returns

The value in host byte order.

Description

Macro used to convert a 16 bit unsigned integer from network to host byte order. It is guaranteed to always return a value of uint16_t (see page 85) type.

2.4.3.33 - Result code messages table definition macros

Macros used to automatically generate messages table initialization for a package.

C++

```
#define MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN(ePkgId)
#define MX_R_PKG_SUCCESS_INFO_MSG_TBL_END(ePkgId)
#define MX_R_PKG_SUCCESS_WARNING_MSG_TBL_BEGIN(ePkgId)
#define MX_R_PKG_SUCCESS_WARNING_MSG_TBL_END(ePkgId)
#define MX_R_PKG_FAIL_ERROR_MSG_TBL_BEGIN(ePkgId)
#define MX_R_PKG_FAIL_ERROR_MSG_TBL_END(ePkgId)
#define MX_R_PKG_FAIL_CRITICAL_MSG_TBL_BEGIN(ePkgId)
#define MX_R_PKG_FAIL_CRITICAL_MSG_TBL_END(ePkgId)
```

Parameters

Parameters	Description
ePkgId	Package ID for which to initialize the message table.

Description

These macros automatically generate messages table initialization for a package. The _MSG_TBL_BEGIN and _MSG_TBL_END macros must be used in pairs. The semicolon must not be used in the messages definition or at the end of the macro. Note that these macros are not defined when the application is compiled without support for error messages (i.e. when MXD_RESULT_ENABLE_ERROR_MESSAGE is not defined).

```
Example (init 3 info-success result code msgs to PKG_PACKAGE:
MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN(eMX_PKG_PACKAGE)
"This SUCCESS INFO result code msg matches with the first result code ID",
"This SUCCESS INFO result code msg matches with the second result code ID",
"This SUCCESS INFO result code msg matches with the third result code ID"
MX_R_PKG_SUCCESS_INFO_MSG_TBL_END(eMX_PKG_PACKAGE)
```

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

mxt_result (see page 92), MX_RGET_PKG_BASE_SUCCESS_WARNING_CODE_ID (see page 49)

2.4.3.34 - Macros to get the base result code id

Macro used to get the lower value of the result code ID enumeration for a unique package ID.

C++

```
#define MX_RGET_PKG_BASE_SUCCESS_WARNING_CODE_ID(ePkgId)
#define MX_RGET_PKG_BASE_FAIL_ERROR_CODE_ID(ePkgId)
#define MX_RGET_PKG_BASE_FAIL_CRITICAL_CODE_ID(ePkgId)
```

Parameters

Parameters	Description
ePkgId	Package ID for which to get the lower value of the result code.

Description

The MX_RGET_PKG_BASE_ macros are used to get the lower value of the result code ID enumeration for a unique package ID. The package ID EMxPackageld (see page 12) must be passed to it so base codes can be unique throughout the system.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

mxt_result (see page 92), MX_PKG_SUCCESS_INFO_MSG_TBL_BEGIN (see page 49)

2.4.3.35 - Result manipulation macros

Result manipulation macro.

C++

```
#define MX_RIS_S(res)
#define MX_RIS_SI(res)
#define MX_RIS_SW(res)
#define MX_RIS_F(res)
#define MX_RIS_FE(res)
#define MX_RIS_FC(res)
#define MX_RGET_LEV(res)
#define MX_RGET_PID(res)
#define MX_RGET_CID(res)
#define MX_RGET_MSG_STR(res)
#define MX_RGET_WORST_OF(resOne, resTwo)
```

Parameters

Parameters	Description
res	Result on which to perform the manipulation.

Description

The following result manipulation macros are available within the framework. Note that if an application wants to use any of these macros, it must include "Config/MxConfig.h" first.

MX_RIS_S

Validation macro. Returns true if the result given in parameter is a success code.

```
bool bValidation = MX_RIS_S(res);
```

MX_RIS_F

Validation macro. Returns true if the result given in parameter is a fail code.

```
bool bValidation = MX_RIS_F(res);
```

MX_RIS_SI

Validation macro. Returns true if the result given in parameter is a success-info code.

```
bool bValidation = MX_RIS_SI(res);
```

MX_RIS_SW

Validation macro. Returns true if the result given in parameter is a success-warning code.

```
bool bValidation = MX_RIS_SW(res);
```

MX_RIS_FE

Validation macro. Returns true if the result given in parameter is a fail-error code.

```
bool bValidation = MX_RIS_FE(res);
```

MX_RIS_FC

Validation macro. Returns true if the result given in parameter is a fail-critical code.

```
bool bValidation = MX_RIS_FC(res);
```

MX_RGET_LEVEL

Macro to get the level of a result.

MX_RGET_PID

Macro to get the package ID of a result.

MX_RGET_CID

Macro to get the code ID of a result.

MX_RGET_MSG_STR

Macro to get the result message string.

MX_RGET_WORST_OF

Macro to get the worst between two results. The comparison is done using the level. If both results have the same level, the first one is always returned.

Notes

The validation macros must never be called to validate the reverse condition. Always use MX_RIS_F instead of !MX_RIS_S. This is to make the code more readable.

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

mxt_result (see page 92)

2.4.3.36 - MX_SIZEOFTARRAY Macro

Returns the size of an array.

C++

```
#define MX_SIZEOFTARRAY(array) (sizeof (array) / sizeof (array[0]))
```

Parameters

Parameters	Description
array	Array for which to get the size.

Description

Returns the size of an array of elements in element count.

2.4.3.37 - MX_SWAPBYTES16 Macro

Swaps the hi and low bytes within a 16 bits unsigned integer.

C++

```
#define MX_SWAPBYTES16(un) (((uint16_t)((uint16_t)un << 8) | ((uint16_t)un >> 8)))
```

Parameters

Parameters	Description
un	16 bits unsigned integer that needs its bytes swapped.

Returns

The bytes swapped resulting 16 bits unsigned integer.

Description

Swaps the hi and low bytes within a 16 bits unsigned integer. 0xaabb becomes 0xbbaa.

2.4.3.38 - MX_SWAPBYTES32 Macro

Swaps the hi and low bytes within a 32 bits unsigned integer.

C++

```
#define MX_SWAPBYTES32(un) (((uint32_t) (((uint32_t)un >> 24) | ((uint32_t)un << 24) | (((uint32_t)un & 0x00ff0000ul) >> 8) | (((uint32_t)un & 0x0000ff00ul) << 8)) )
```

Parameters

Parameters	Description
un	32 bits unsigned integer that needs its bytes swapped.

Returns

The bytes swapped resulting 32 bits unsigned integer.

Description

Swaps all 4 bytes within a 32 bits unsigned integer. 0xaabbccdd becomes 0xddccbbaa.

2.4.3.39 - Tracing macros

Standard tracing macros.

C++

```
#define MX_TRACE(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE0(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE1(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE2(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE3(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE4(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE5(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE6(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE7(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE8(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)  
#define MX_TRACE9(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const char* pszMsgFormat, ...)
```

Parameters

Parameters	Description
pszMsgFormat	The trace message format as in the format argument of printf.
va_args	The variable arguments corresponding to the specifier in pszMsgFormat as with printf.
pstNode	The node associated with the trace.
eLevel	The level to use to issue the trace. Only present in MX_TRACE.
uTraceUniqueId	The unique ID associated with the trace.

Description

This is the standard tracing macro. One different macro exists for each tracing level from 0 to 9. When a level is enabled at compile time with the corresponding pre-processor define (MXD_TRACE0_ENABLE_SUPPORT (see page 314)), if that level is enabled at run time and the node pointed by pstNode is enabled, the trace is sent to the format handler and then to the output handler.

Also, when a level is enabled at compile time, the parameters passed to the macro are evaluated and the associated code is part of the code size. On the other hand, when the level is disabled, the trace is taken out from the program by the compiler and there is no code size penalty for the trace.

There is a particular case for the macro using eLevel as its first parameter. This macro is useful but only in rare circumstances. It must be used sparingly as it cannot be taken out by the compiler even if the level passed in eLevel is disabled, except if the entire tracing mechanism is disabled.

Typical usage is shown in the following code example:

```

// Example feature initialization code
STraceNode g_stExampleNode;
MxTraceRegisterNode (&g_stTraceRoot, &g_stExampleNode, "TEST_NODE");

...
// Example feature execution
MX_TRACE0(0,&g_stExampleNode,"Trace situation 1");

unsigned int uIt = 0;
for (uIt = 0; uIt < 10; uIt++)
{
    MX_TRACE7(0,&g_stExampleNode,"Trace situation 2 iteration %d", uIt);
}

...
// Example feature finalization code
STraceNode g_stExampleNode;
MxTraceUnregisterNode (&g_stTraceRoot, &g_stExampleNode);

```

See Also

General Tracing Configuration (see page 2)

2.4.3.40 - Call stack tracing macros

Call stack tracing macros.

C++

```

#define MX_TRACE_CALLSTACK(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN STraceNode* pstNode)
#define MX_TRACE_ALL_CALLSTACKS(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN STraceNode* pstNode)
#define MX_TRACE_CALLSTACK_COREDUMP(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN STraceNode* pstNode,
IN void *p1)

```

Parameters

Parameters	Description
pstNode	The node associated with the trace.
p1	Always set to NULL. Only present in MX_TRACE_CALLSTACK_COREDUMP.
eLevel	The level to use to issue the trace.
uTraceUniqueId	The unique ID associated with the trace.

Description

These special tracing macros produce (for a limited OS/Architecture combination) a series of traces showing the function call stack. The actual trace contains the appropriate function name and/or address, as in the example below.

```

Start dumping call stack
Build/TestFw21.exe [0x10075370]
Build/TestFw21.exe [0x10078210]
Build/TestFw21.exe [0x10076c8c]
Build/TestFw21.exe [0x10077374]
Build/TestFw21.exe(main+0x188) [0x10077b6c]
/1ib/libc.so.6(__libc_start_main+0x144) [0xfd35be4]
End dumping call stack

```

The tracing function behind this macro might not be able to print all the function names. You may need to have the map file in hand (and/or the program disassembly) to interpret the call stack trace.

WARNING:

Calling MX_TRACE_CALLSTACK may slow the execution of your code and change the application timing. It should only be used in appropriate situations.

MX_TRACE_CALLSTACK shows the call stack of the thread currently executing.

MX_TRACE_ALL_CALLSTACK shows the call stack of all the threads currently in the system.

MX_TRACE_CALLSTACK_COREDUMP shows the state of the registers and the call stack of the thread currently executing.

See Also

General Tracing Configuration (see page 2)

2.4.3.41 - Hexadecimal tracing macros

Hexadecimal tracing macros.

C++

```
#define MX_TRACE_HEX(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const
uint8_t* puIn, IN unsigned int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE0_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE1_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE2_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE3_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE4_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE5_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE6_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE7_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE8_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
#define MX_TRACE9_HEX(IN uint32_t uTraceUniqueId, IN STraceNode* pstNode, IN const uint8_t* puIn, IN unsigned
int uInSize, IN const char* pszMsgFormat, ...)
```

Parameters

Parameters	Description
uInSize	The size, in bytes, of the buffer pointed by puIn.
pszMsgFormat	The trace message format as in the format argument of printf.
va_args	The variable arguments corresponding to the specifier in pszMsgFormat as with printf.
puIn	A pointer to the begining of hexadecimal data to trace.
eLevel	The level to use to issue the trace. Only present in MX_TRACE_HEX.
uTraceUniqueId	The unique ID associated with the trace.
pstNode	The node associated with the trace.

Description

This is the hexadecimal tracing macro. One different macro exists for each tracing level from 0 to 9. The hexadecimal macros follow the same rules as the MX_TRACE (see page 52) macros with the exception they trace additional uInSize bytes of hexadecimal data pointed by puIn.

Typical usage is shown in the following code example:

```
// Example feature initialization code
STraceNode g_stExampleNode;
MxTraceRegisterNode(&g_stTraceRoot, &g_stExampleNode, "TEST_NODE");

...

// Example feature execution
uint32_t auHexData[] = {1,2,3,4};
MX_TRACE2_HEX(0, &g_stExampleNode, auHexData, sizeof(auHexData), "Trace situation 1");

unsigned int uIt = 0;
for (uIt = 0; uIt < 10; uIt++)
{
    MX_TRACE8_HEX(0, &g_stExampleNode, auHexData, sizeof(auHexData), "Trace situation 2 iteration %d", uIt);
}

...

// Example feature finalization code
STraceNode g_stExampleNode;
MxTraceUnregisterNode (&g_stTraceRoot, &g_stExampleNode);
```

See Also

General Tracing Configuration (see page 2)

2.4.3.42 - Tracing level activation check macros

Tracing level activation check macros.

C++

```
#define MX_TRACE0_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE1_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE2_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE3_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE4_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE5_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE6_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE7_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE8_IS_ENABLED(STRaceNode* pstNode)
#define MX_TRACE9_IS_ENABLED(STRaceNode* pstNode)
```

Parameters

Parameters	Description
pstNode	A pointer to the node to evaluate.

Returns

True if the level and the node are enabled, false otherwise.

Description

This is a helper tracing macro. One different macro exists for each tracing level from 0 to 9. When a level is enabled at compile time with the corresponding pre-processor define (MXD_TRACE0_ENABLE_SUPPORT (see page 314)), if that level is enabled at run time and the node pointed by pstNode is enabled, the macro returns true.

This macro can be very helpful because it gives the possibility to validate the condition used to determine if the trace is outputted before calling a tracing macro. In the case it is not outputted, the processing needed to prepare the data to trace can be skipped.

When a level is disabled at compile time, some compiler might even take the code inside the "if" statement out from the object code when compiling with the right optimization level.

Typical usage is shown in the following code example:

```
// Example feature initialization code
STRaceNode g_stExampleNode;
MxTraceRegisterNode (&g_stTraceRoot, &g_stExampleNode, "TEST_NODE");

...
// Example feature execution
MX_TRACE0(0, &g_stExampleNode, "Trace situation 1");

if (MX_TRACE7_IS_ENABLED(g_stExampleNode))
{
    // Decrypt buffer to access text data that is needed to
    // issue the trace. The decryption is costly, we want to
    // do it only if the trace will be outputted because it
    // is not part of normal processing.
    Decrypt(puCipherText, pszPlainText);

    MX_TRACE7(0, &g_stExampleNode, "Trace encrypted text: %s", pszPlainText);
}

...
// Example feature finalization code
STRaceNode g_stExampleNode;
MxTraceUnregisterNode (&g_stTraceRoot, &g_stExampleNode);
```

See Also

General Tracing Configuration (see page 2)

2.4.3.43 - MX_PACK MACROS

Macro for packing structures and class members.

Description

Macro for packing class members. MX_PACK_MEMBER Macro for packing structure. MX_PACK_STRUCT

The packing and alignment attributes are usually set to the compiler default values. Therefore, when it is required that a structure or class is packed or aligned using values different from the default values, it must be specified explicitly using compiler specific keywords. It is assumed that when changed for a specific definition, these attributes are set back to the compiler defaults.

This mechanism is intended to always pack on the smallest possible alignment (1 byte for variable, 1 bit for bit-field). The mechanism

follows the assumption described in the previous paragraph by reverting the attributes to their default value.

Examples of good utilization:

```

1- #include "Basic/MxPack.h"
struct SSomeStruct
{
    char cMember      MX_PACK_MEMBER;
    int16_t nMember   MX_PACK_MEMBER;
    uint32_t uMember  MX_PACK_MEMBER;
} MX_PACK_STRUCT;
#include "Basic/MxPack.h"

2- #include "Basic/MxPack.h"
struct SSomeStruct
{
    char cMember      MX_PACK_MEMBER;
    int16_t nMember   MX_PACK_MEMBER;
    uint32_t uMember  MX_PACK_MEMBER;
} MX_PACK_STRUCT;

struct SSomeOtherStruct
{
    char cMember      MX_PACK_MEMBER;
    int16_t nMember   MX_PACK_MEMBER;
    uint32_t uMember  MX_PACK_MEMBER;
} MX_PACK_STRUCT;
#include "Basic/MxPack.h"

3- #include "SomeOtherStruct.h"
#include "Basic/MxPack.h"
struct SSomeStruct
{
    char cMember      MX_PACK_MEMBER;
    int16_t nMember   MX_PACK_MEMBER;
    uint32_t uMember  MX_PACK_MEMBER;
} MX_PACK_STRUCT;
#include "Basic/MxPack.h"

```

Caveats:

- Bit-fields are not supported when MXD_COMPILER_DIAB (see page 278) is defined.
- Never include a file between the two MxPack includes to avoid unwanted side effects. The side effects are related to compiler specific primitives. For example, in the following code, only the SSomeOtherStruct structure is packed when MXD_COMPILER_DIAB (see page 278) or MXD_COMPILER_MS_VC (see page 279) is defined. On the other hand, there are no side effects when MXD_COMPILER_GNU_GCC (see page 279) is defined.

```

Example possibly introducing side effects:
SomeStruct.h:
    #include "Basic/MxPack.h"
    struct SSomeStruct
    {
        char cMember      MX_PACK_MEMBER;
        int16_t nMember   MX_PACK_MEMBER;
        uint32_t uMember  MX_PACK_MEMBER;
    };
#include "Basic/MxPack.h"
SomeOtherStruct.h:
    #include "Basic/MxPack.h"
    #include "SomeStruct.h"
    struct SSomeOtherStruct
    {
        char cMember      MX_PACK_MEMBER;
        int16_t nMember   MX_PACK_MEMBER;
        uint32_t uMember  MX_PACK_MEMBER;
    };
#include "Basic/MxPack.h"

```

2.4.3.44 - MXD_GNS

Defines the global namespace.

Description

When namespacing is enabled, defines the name assigned to the namespace of this product. Defaults to m5t.

Location

Define this in "PreMxConfig.h" to change the default value.

See Also

[MX_NAMESPACE_START](#) (see page 47), [MX_NAMESPACE_END](#) (see page 47), [MX_NAMESPACE_USE](#) (see page 48)

2.4.3.45 - MX_MAKEUINT64 Macro

Makes a 64 bits unsigned integer from two 32 bits, hi and low portions.

C++

```
#define MX_MAKEUINT64(high, low) static_cast<uint64_t>(<static_cast<uint64_t>(high) << 32 |<static_cast<uint32_t>(low) >> 32>)
```

Parameters

Parameters	Description
high	Higher 32 bits of the unsigned integer to define.
low	Lower 32 bits of the unsigned integer to define.

Description

Makes a 64 bits unsigned integer from two 32 bits, hi and low portions.

See Also

[MX_MAKEUINT16](#) (see page 46), [MX_MAKEUINT32](#) (see page 46)

2.4.3.46 - MX_HI32 Macro

Gets the hi 32 bits portion of an integer.

C++

```
#define MX_HI32(n) (static_cast<uint32_t>((static_cast<uint64_t>(n) >> 32)))
```

Parameters

Parameters	Description
n	Integer from which to get the highest 32 bits.

Description

Gets the hi 32 bits portion of an integer.

See Also

[MX_LOW8](#) (see page 45), [MX_LOW16](#) (see page 45), [MX_LOW32](#) (see page 45), [MX_HI8](#) (see page 44), [MX_HI16](#) (see page 43)

2.4.3.47 - MX_SWAPBYTES64 Macro

Swaps the hi and low bytes within a 64 bits unsigned integer.

C++

```
#define MX_SWAPBYTES64(un) (((uint64_t)un >> 56) | ((uint64_t)un << 56) | (((uint64_t)un & MX_MAKEUINT64(0x00ff0000,0x00000000)) >> 40) | (((uint64_t)un & MX_MAKEUINT64(0x00000000,0x0000ff00)) << 40) | (((uint64_t)un & MX_MAKEUINT64(0x0000ff00,0x00000000)) >> 24) | (((uint64_t)un & MX_MAKEUINT64(0x00000000,0x00ff0000)) << 24) | (((uint64_t)un & MX_MAKEUINT64(0x000000ff,0x00000000)) >> 8) | (((uint64_t)un & MX_MAKEUINT64(0x00000000,0xff000000)) << 8) ) )
```

Parameters

Parameters	Description
un	64 bits unsigned integer that needs its bytes swapped.

Returns

The bytes swapped resulting 64 bits unsigned integer.

Description

Swaps all 8 bytes within a 64 bits unsigned integer. 0x11223344aabbccdd becomes 0xddccbbaa44223311.

2.4.3.48 - MX_ERRNO_SET Macro

Sets the errno value.

C++

```
#define MX_ERRNO_SET(ErrorCode) (errno = (ErrorCode))
```

Parameters

Parameters	Description
ErrorCode	Value to set.

Description

Sets the errno value.

2.4.3.49 - MX_ERRNO_GET Macro

Gets the errno value.

C++

```
#define MX_ERRNO_GET (errno)
```

Description

Gets the errno value.

2.4.3.50 - MX_REMOVE_UNUSED_FUNCTION_PARAM_WARNING Macro

Removes warning about unused function parameters.

C++

```
#define MX_REMOVE_UNUSED_FUNCTION_PARAM_WARNING(x) MxRemoveUnusedFunctionParamWarning((void*)&(x));
```

Parameters

Parameters	Description
rParam	Unused parameter for which to remove warning.

Returns

Returns a constant reference on the parameter itself.

Description

Macro used to avoid the "unused function parameter" compiler warning. In optimized mode, this macro does not generate any code.

2.4.3.51 - MX_MAKE_STRING_NULL_SAFE Macro

Converts a NULL string to its system-dependent equivalent.

C++

```
#define MX_MAKE_STRING_NULL_SAFE(pszValue) pszValue
```

Parameters

Parameters	Description
pszValue	The string to make null safe.

Returns

If pszValue is non-NULL, the current value of pszValue. If pszValue is NULL, the special string g_szNULL (see page 94).

Description

This macro converts a NULL string to its system-dependent equivalent. It is useful mainly for tracing and for calls to system run-time libraries. It helps the developer to prevent crashes caused by NULL strings.

```
void PrintStringToStdout(const char* pszStringToPrint)
{
    // pszStringToPrint comes from an unknown source.
    // Make sure it will not crash the system on platforms that do not
    // check for NULL string in the run-time libraries (i.e. Symbian).
    printf("%s", MX_MAKE_STRING_NULL_SAFE(pszStringToPrint));
}
```

2.4.3.52 - MX_MAKE_NULL_EMPTY_STRING Macro

Converts a NULL string to an empty string.

C++

```
#define MX_MAKE_NULL_EMPTY_STRING(pszValue) (pszValue != NULL ? pszValue : g_szEMPTY_STRING)
```

Parameters

Parameters	Description
pszValue	The string to make NULL safe.

Returns

If pszValue is non-NULL, the current value of pszValue. If pszValue is NULL, an empty string (g_szEMPTY_STRING (see page 94)).

Description

This macro converts a NULL string to a NULL string (i.e. ""). It is useful mainly FOR C-style string comparison.

It helps the developer to prevent crashes caused by NULL strings.

```
int CompareStrings(const char* pszString1,
                   const char* pszString2)
{
    // pszString1 and pszString2 come from an unknown source.

    // Make sure that it will not crash the system. Consider "" equivalent
    // to NULL.
    return strcmp(MX_MAKE_NULL_EMPTY_STRING(pszString1),
                  MX_MAKE_NULL_EMPTY_STRING(pszString2));
}
```

2.4.3.53 - MX_ALIGNMENT_OF Macro

Returns a value, of type size_t, that is the alignment requirement of the given type.

C++

```
#define MX_ALIGNMENT_OF(x) ((unsigned int) __alignof__(x))
```

Description

Returns a value, of type size_t, that is the alignment requirement of the given type.

Notes

Some older compilers, namely MSVC 6.0, do not support getting the alignment from a variable instance, they only support getting the alignment from a type.

2.4.3.54 - MX_VOIDPTR_TO_OPQ Macro

Converts a void pointer to an opaque (mxt_opaque (see page 87)).

C++

```
#define MX_VOIDPTR_TO_OPQ(pParam) reinterpret_cast<mxt_opaque>(pParam)
```

Parameters

Parameters	Description
pParam	Pointer to convert.

Returns

The converted value to opaque.

Description

Converts a void pointer to an opaque (mxt_opaque (see page 87)). The usage of this macro is mandatory as shown in the following example:

```
CObject* pObject = MX_NEW(CObject);
mxt_opaque opq = MX_VOIDPTR_TO_OPQ(pObject);
CObject* pObject2 = reinterpret_cast<CObject*>(MX_OPQ_TO_VOIDPTR(opq));
```

See Also

mxt_opaque (see page 87), MX_OPQ_TO_VOIDPTR (see page 59)

2.4.3.55 - MX_OPQ_TO_VOIDPTR Macro

Converts an opaque (mxt_opaque (see page 87)) to a void pointer.

C++

```
#define MX_OPQ_TO_VOIDPTR(opq) reinterpret_cast<void*>(opq)
```

Parameters

Parameters	Description
opq	opaque to convert.

Returns

The converted pointer value.

Description

Converts an opaque (mxt_opaque (see page 87)) to a void pointer. The usage of this macro is mandatory as shown in the following example:

```
CObject* pObject = MX_NEW(CObject);
mxt_opaque opq = MX_VOIDPTR_TO_OPQ(pObject);
CObject* pObject2 = reinterpret_cast<CObject*>(MX_OPQ_TO_VOIDPTR(opq));
```

See Also

[mxt_opaque](#) (see page 87), [MX_VOIDPTR_TO_OPQ](#) (see page 59)

2.4.3.56 - MX_INT32_TO_OPQ Macro

Converts a 32 bits signed integer to an opaque (mxt_opaque (see page 87)).

C++

```
#define MX_INT32_TO_OPQ(nParam) reinterpret_cast<mxt_opaque>(nParam)
```

Parameters

Parameters	Description
nParam	32 bits signed integer to convert.

Returns

The converted value to opaque.

Description

Converts a 32 bits signed integer to an opaque (mxt_opaque (see page 87)).

The mxt_opaque (see page 87) is defined as a void pointer. This results in a 64 bits type on a 64 bits platform and in a 32 bits type on a 32 bits platform. To ease the Framework portability on any 32 and 64 bits platforms and for better performance, only 32 bits integer may be converted to opaque. This is also true in a 64 bits platform.

The usage of this macro is mandatory as shown in the following example:

```
int nNumber = 0;
mxt_opaque opq = MX_INT32_TO_OPQ(nNumber);
int nNumber2 = MX_OPQ_TO_INT32(opq);
```

See Also

[mxt_opaque](#) (see page 87), [MX_OPQ_TO_INT32](#) (see page 60)

2.4.3.57 - MX_OPQ_TO_INT32 Macro

Converts an opaque (mxt_opaque (see page 87)) to a 32 bits signed integer.

C++

```
#define MX_OPQ_TO_INT32(opq) static_cast<int32_t>(reinterpret_cast<size_t>(opq))
```

Parameters

Parameters	Description
opq	opaque to convert.

Returns

The converted pointer value.

Description

Converts an opaque (mxt_opaque (see page 87)) to a 32 bits signed integer.

The usage of this macro is mandatory as shown in the following example:

```
int nNumber = 0;
mxt_opaque opq = MX_UINT32_TO_OPQ(nNumber);
int nNumber2 = MX_OPQ_TO_INT32(opq);
```

See Also

[mxt_opaque](#) (see page 87), [MX_UINT32_TO_OPQ](#) (see page 60)

2.4.3.58 - MX_UINT32_TO_OPQ Macro

Converts a 32 bits unsigned integer to an opaque (mxt_opaque (see page 87)).

C++

```
#define MX_UINT32_TO_OPQ(uParam) reinterpret_cast<mxt_opaque>(uParam)
```

Parameters

Parameters	Description
uParam	32 bits unsigned integer to convert.

Returns

The converted value to opaque.

Description

Converts a 32 bits unsigned integer to an opaque (mxt_opaque (see page 87)).

The mxt_opaque (see page 87) is defined as a void pointer. This results in a 64 bits type on a 64 bits platform and in a 32 bits type on a 32 bits platform. To ease the Framework portability on any 32 and 64 bits platforms and for better performance, only 32 bits integer may be converted to opaque. This is also true in a 64 bits platform.

The usage of this macro is mandatory as shown in the following example:

```
unsigned int uNumber = 0;
mxt_opaque opq = MX_UINT32_TO_OPQ(uNumber);
unsigned int uNumber2 = MX_OPQ_TO_UINT32(opq);
```

See Also

[mxt_opaque](#) (see page 87), [MX_OPQ_TO_UINT32](#) (see page 61)

2.4.3.59 - MX_OPQ_TO_UINT32 Macro

Converts an opaque (mxt_opaque (see page 87)) to a 32 bits unsigned integer.

C++

```
#define MX_OPQ_TO_UINT32(opq) static_cast<uint32_t>(reinterpret_cast<size_t>(opq))
```

Parameters

Parameters	Description
opq	opaque to convert.

Returns

The converted pointer value.

Description

Converts an opaque (mxt_opaque (see page 87)) to a 32 bits unsigned integer.

The usage of this macro is mandatory as shown in the following example:

```
unsigned int uNumber = 0;
mxt_opaque opq = MX_UINT32_TO_OPQ(uNumber);
unsigned int uNumber2 = MX_OPQ_TO_UINT32(opq);
```

See Also

[mxt_opaque](#) (see page 87), [MX_UINT32_TO_OPQ](#) (see page 61)

2.4.3.60 - MX_ISDIGIT Macro

Verifies if a character is alphabetic or numeric.

C++

```
#define MX_ISDIGIT(cCharacter) isdigit(static_cast<unsigned char>(cCharacter))
```

Parameters

Parameters	Description
cCharacter	character to verify.

Returns

- 0: cCharacter is neither alphabetic nor numeric.
- different than 0: cCharacter is a lowercase or uppercase alphabetic character or a number.

Description

Verifies if a character is a lowercase or uppercase alphabetic character or a number.

See Also

MX_ISDIGIT, MX_ISALPHA

2.4.4 - Deprecated Macros

This section documents the deprecated macros of the Sources/Basic folder.

Macros

Macro	Description
TO (see page 62)	Defines parameter direction and ownership.

2.4.4.1 - Deprecated parameters qualifier

Defines parameter direction and ownership.

C++

```
#define TO
```

Description

Modifiers help to eliminate confusion around parameter direction and ownership acquisition. The modifiers for direction and ownership can be combined. Modifiers are used in the member function signature but may also be used in function call to eliminate ambiguity.

TO:

Indicates that the ownership of the pointer is taken from the caller. The caller is no longer responsible for the object deletion. The TO Macro is deprecated since 2.1.4. TOA (see page 39) (Take Ownership Always) and TOS (see page 39) (Take Ownership Success) are now used instead.

See Also

Parameters qualifier (see page 39)

2.4.5 - Structures

This section documents the structures of the Sources/Basic folder.

Structs

Struct	Description
SAssertCallStackTraceHandler (see page 63)	Defines the structure to be given back to the callstack trace handler.
SAssertFailHandler (see page 63)	Defines the structure to be given back to the assertion handler.
SAssertFinalBehaviorHandler (see page 63)	Defines the structure to be given back to the final behaviour handler.
SAssertTraceHandler (see page 63)	Defines the structure to be given back to the trace handler.
STraceNode (see page 64)	Defines the structure of the tracing tree node.

2.4.5.1 - SAssertCallStackTraceHandler Struct

Defines the structure to be given back to the callstack trace handler.

C++

```
typedef struct {
    mxt_PFNAssertCallStackTraceHandler pfnHandler;
    mxt_opaque opq;
} SAssertCallStackTraceHandler;
```

Description

Defines the structure that associates a prototype for an assertion call stack trace handler and an opaque value that is given back to the handler when it is called.

Members

Members	Description
mxt_PFNAssertCallStackTraceHandler pfnHandler;	Pointer to an assertion call stack trace handler function.
mxt_opaque opq;	Opaque value to be passed to the assertion call stack trace handler function.

2.4.5.2 - SAssertFailHandler Struct

Defines the structure to be given back to the assertion handler.

C++

```
typedef struct {
    mxt_PFNAssertFailHandler pfnHandler;
    mxt_opaque opq;
} SAssertFailHandler;
```

Description

Defines the structure that associates a prototype for an assertion failed handler and an opaque value that is given back to the handler when it is called.

Members

Members	Description
mxt_PFNAssertFailHandler pfnHandler;	Pointer to an assertion failed handler function.
mxt_opaque opq;	Opaque value to be passed to the assertion failed handler function.

2.4.5.3 - SAssertFinalBehaviorHandler Struct

Defines the structure to be given back to the final behaviour handler.

C++

```
typedef struct {
    mxt_PFNAssertFinalBehaviorHandler pfnHandler;
    mxt_opaque opq;
} SAssertFinalBehaviorHandler;
```

Description

Defines the structure that associates a prototype for an assertion final behaviour handler and an opaque value that is given back to the handler when it is called.

Members

Members	Description
mxt_PFNAssertFinalBehaviorHandler pfnHandler;	Pointer to an assertion final behavior handler function.
mxt_opaque opq;	Opaque value to be passed to the assertion final behavior handler function.

2.4.5.4 - SAssertTraceHandler Struct

Defines the structure to be given back to the trace handler.

C++

```
typedef struct {
    mxt_PFNAssertTraceHandler pfnHandler;
    mxt_opaque opq;
} SAssertTraceHandler;
```

Description

Defines the structure that associates a prototype for an assertion trace handler and an opaque value that is given back to the handler when it is called.

Members

Members	Description
<code>mxt_PFNAssertTraceHandler pfnHandler;</code>	Pointer to an assertion trace handler function.
<code>mxt_opaque opq;</code>	Opaque value to be passed to the assertion trace handler function.

2.4.5.5 - STraceNode Struct

Defines the structure of the tracing tree node.

C++

```
struct STraceNode {
    bool m_bEnabled;
    EMxTraceLevel m_eLevelEnabled;
    struct STraceNode* m_pstNextNode;
    struct STraceNode* m_pstChildNode;
    const char* m_pszName;
};
```

Description

Defines the structure that is used to build the tracing tree. This tree is used to show/hide traces according to the user's configuration. A tracing node must be initialized and registered using the MxTraceRegisterNode (see page 30) method.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

See Also

[MxTraceRegisterNode](#) (see page 30), [MxTraceUnregisterNode](#) (see page 34), [MxTraceEnableNode](#) (see page 27), [MxTraceDisableNode](#) (see page 24)

Members

Members	Description
<code>bool m_bEnabled;</code>	The node state.
<code>EMxTraceLevel m_eLevelEnabled;</code>	Node specific tracing level.
<code>struct STraceNode* m_pstNextNode;</code>	The next node with same level.
<code>struct STraceNode* m_pstChildNode;</code>	The child node.
<code>const char* m_pszName;</code>	The node name.

2.4.6 - Templates

This section documents the templates of the Sources/Basic folder.

Templates

Template	Description
<code>CAignedVariableStorage</code> (see page 64)	Implements a <code>uint32_t</code> (see page 85) (or a <code>uint64_t</code> (see page 85) for Solaris) aligned container.
<code>CAutoPtr</code> (see page 68)	Class that automatically manages an object's lifetime.
<code>CSharedPtr</code> (see page 75)	Class that automatically manages an object's lifetime.

2.4.6.1 - CAignedVariableStorage Template

Implements a `uint32_t` (see page 85) (or a `uint64_t` (see page 85) for Solaris) aligned container.

Class Hierarchy

`CAignedVariableStorage`

C++

```
template <class _Type>
class CAignedVariableStorage;
```

Description

The CAignedVariableStorage class provides a uint32_t (see page 85) (or a uint64_t (see page 85) for Solaris) aligned container for objects that require such an alignment.

The most common scenario is when an object is allocated into a byte array using a placement new. Depending on the platform, the byte array may not be aligned properly. As a result, if the allocated object needs to access a member that requires alignment, it will fail. The failure may cause a crash, a processor exception, a bus error, etc.

The following is a simple example of an encountered problem.

```
class CExample
{
public:
    CExample();
    ~CExample();
    void*      m_pVoid;
};

uint8_t gs_auArray[sizeof(CExample)];
int main()
{
    CExample* pExample = new (gs_auArray) CExample;
    // ****
    // Unaligned access will occur if auArray is not aligned.
    // ****
    pExample->m_pVoid = NULL;

    // Object destruction.
    // -----
    reinterpret_cast<CExample*>(gs_auArray)->~CExample();
    ...
}
```

To ensure that the allocated object is properly aligned, the CAignedVariableObject must be used as below:

```
CAlignedVariableStorage<CExample> gs_example;
int main()
{
    // Object construction.
    // -----
    CExample* pExample = gs_example.Construct();

    // ****
    // Access will always be aligned.
    // ****
    pExample->m_pVoid = NULL;

    // Object destruction.
    // -----
    pExample.Destruct();
}
```

Additional note: If MXD_DEFAULT_ALIGNMENT_TYPE is defined, the type it defines is then used to align the structure. Otherwise, the default alignment of _Type is used,

Warning

Using this template requires that instantiated objects have a default constructor, a destructor, and at least one non static data member.

Location

Basic/CAignedVariableStorage.h

Constructors

Constructor	Description
CAignedVariableStorage (see page 66)	Empty constructor.

Legend

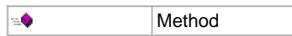
	Method
---	--------

Destructors

Destructor	Description
~CAignedVariableStorage (see page 66)	Empty destructor. The destructor is wanted to be non virtual for two reasons. First, there are no foreseen usecases where a class will inherit from this one. Finally, the object size will be kept to the minimum since there will be no virtual table.

Legend**Operators**

Operator	Description
<code>~_Type&</code> (see page 67)	Returns a reference to the object's value.
<code>~_Type =</code> (see page 68)	Assigns a value to the object.

Legend**Methods**

Method	Description
<code>~_Type Construct</code> (see page 66)	Constructs the object and returns an instance pointer.
<code>~_Type Destruct</code> (see page 67)	Destructs the object.
<code>~_Type Instance</code> (see page 67)	Returns a pointer to the object's instance.

Legend**2.4.6.1.1 - Constructors****2.4.6.1.1.1 - CAignedVariableStorage::CAignedVariableStorage Constructor**

Empty constructor.

C++

```
CAignedVariableStorage();
```

2.4.6.1.2 - Destructors**2.4.6.1.2.1 - CAignedVariableStorage::~CAignedVariableStorage Destructor**

Empty destructor. The destructor is wanted to be non virtual for two reasons. First, there are no foreseen usecases where a class will inherit from this one. Finally, the object size will be kept to the minimum since there will be no virtual table.

C++

```
~CAignedVariableStorage();
```

2.4.6.1.3 - Methods**2.4.6.1.3.1 - CAignedVariableStorage::Construct Method**

Constructs the object and returns an instance pointer.

C++

```
_Type* Construct();
```

Returns

A pointer to the object's instance.

Description

Constructs the object using a placement new. The object's constructor is invoked. It also returns a pointer to the created object's instance.

Warning

Using this template requires that instantiated objects have a default constructor, a destructor, and at least one non static data member.

See Also

[Destruct](#) (see page 67)

2.4.6.1.3.2 - CAignedVariableStorage::Destruct Method

Destructs the object.

C++

```
void Destruct();
```

Description

Destructs the object. The object's destructor is invoked.

Warning

Using this template requires that instantiated objects have a default constructor, a destructor, and at least one non static data member.

See Also

Construct (see page 66)

2.4.6.1.3.3 - Instance

2.4.6.1.3.3.1 - CAignedVariableStorage::Instance Method

Returns a pointer to the object's instance.

C++

```
_Type* Instance();
```

Returns

Returns a pointer to the object's instance.

Description

Returns the object's instance.

2.4.6.1.3.3.2 - CAignedVariableStorage::Instance Method

Returns a const pointer to the object's instance.

C++

```
const _Type* Instance() const;
```

Returns

Returns a const pointer to the object's instance.

Description

Returns the object's instance.

2.4.6.1.4 - Operators

2.4.6.1.4.1 - CAignedVariableStorage::_Type& Operator

Returns a reference to the object's value.

C++

```
operator _Type&();
```

Returns

A reference to the object's value.

Description

Returns a reference to the object's value.

2.4.6.1.4.2 - CAignedVariableStorage::= Operator

Assigns a value to the object.

C++

```
CAignedVariableStorage<_Type>& operator =(IN const _Type& rValue);
```

Parameters

Parameters	Description
IN const _Type& rValue	Object's value.

Returns

A reference to this instance.

Description

Assigns a value to the object.

2.4.6.2 - CAutoPtr Template

Class that automatically manages an object's lifetime.

Class Hierarchy

CAutoPtr

C++

```
template <class _Type>
class CAutoPtr;
```

Description

The CAutoPtr class automatically manages an object's lifetime. It automatically deletes from memory the object it has been assigned once the CAutoPtr object is destroyed. This effectively means that an object allocated in memory and put in a CAutoPtr never has to be deleted by using MX_DELETE (see page 509), thus reducing the risks of memory leaks.

The CAutoPtr class is especially useful when declared on the stack or as a class member since it automatically deletes the managed object once it is destroyed.

Location

Basic/CAutoPtr.h

Example

In the code below, an object is created in the heap. That object is automatically put in the CAutoPtr object that manages its lifetime. The object is automatically released once the CAutoPtr object is deleted. In this example, this occurs when the function returns.

```
void function()
{
    CAutoPtr<CClass> spClass(MX_NEW(CClass));
    spClass->ClassMethod();
}
```

Constructors

Constructor	Description
CAutoPtr (see page 69)	Copy constructor.

Legend

	Constructor
	Method

Destructors

Destructor	Description
~CAutoPtr (see page 70)	Destructor.

Legend

	Destructor
	Method

Operators

Operator	Description
<code>!=</code> (see page 71)	Different than operator.
<code>*</code> (see page 71)	Reference operator.
<code><</code> (see page 72)	Less than operator.
<code><=</code> (see page 72)	Less than or equal operator.
<code>=</code> (see page 72)	Assignment operator.
<code>==</code> (see page 73)	Comparison operator.
<code>></code> (see page 73)	Greater than operator.
<code>-></code> (see page 74)	Dereference operator.
<code>>=</code> (see page 74)	Greater than or equal operator.

Legend

	Method
--	--------

Methods

Method	Description
<code>Get</code> (see page 70)	Gets the pointer to the object.
<code>Release</code> (see page 70)	Releases the object.
<code>Reset</code> (see page 71)	Resets the object.

Legend

	Method
--	--------

2.4.6.2.1 - Constructors

2.4.6.2.1.1 - CAutoPtr

2.4.6.2.1.1.1 - CAutoPtr::CAutoPtr Constructor

Copy constructor.

C++

```
CAutoPtr(IN TOA CAutoPtr<_Type>& rSrc);
```

Parameters

Parameters	Description
IN TOA CAutoPtr<_Type>& rSrc	A reference to the CAutoPtr object to copy.

Description

This is the copy constructor of the CAutoPtr. It creates a new CAutoPtr object that manages a new reference of the object.

See Also

`~CAutoPtr`

2.4.6.2.1.1.2 - CAutoPtr::CAutoPtr Constructor

Default constructor.

C++

```
explicit CAutoPtr(TOA _Type* pObject = NULL);
```

Parameters

Parameters	Description
TOA _Type* pObject = NULL	Pointer to an object to be managed by this CAutoPtr. NULL by default.

Description

This is the default constructor of the CAutoPtr object. If `pObject` is not NULL, this object takes its ownership.

See Also[~CAutoPtr](#)**Example**

This example shows different ways to create a CAutoPtr object using its default constructor.

```
CAutoPtr<CClass> spClass;
CClass* pClass;
CAutoPtr<CClass> spClass(TOA pClass);
pClass = NULL;
```

2.4.6.2.2 - Constructors**2.4.6.2.2.1 - CAutoPtr::~CAutoPtr Destructor**

Destructor.

C++

```
~CAutoPtr( );
```

Description

This is the destructor of the CAutoPtr (see page 68) object. It releases the reference on the object it manages.

See Also[CAutoPtr](#) (see page 68)**2.4.6.2.3 - Methods****2.4.6.2.3.1 - CAutoPtr::Get Method**

Gets the pointer to the object.

C++

```
_Type* Get( ) const;
```

Returns

- A pointer to the object managed by this CAutoPtr (see page 68).

Description

Returns a pointer to the object managed by this object.

See Also[Release](#) (see page 70)**2.4.6.2.3.2 - CAutoPtr::Release Method**

Releases the object.

C++

```
_Type* GO Release( );
```

Returns

- The reference to the object managed by this CAutoPtr (see page 68).

Description

Releases the object managed by this object. A pointer to the object is returned, and this CAutoPtr (see page 68) stops managing the object, meaning that it no longer has a pointer to the object. Ownership of the object is given to the caller of the method.

See Also[Get](#) (see page 70)

2.4.6.2.3.3 - CAutoPtr::Reset Method

Resets the object.

C++

```
void Reset(IN TOA _Type* pObject = NULL);
```

Parameters

Parameters	Description
IN TOA _Type* pObject = NULL	A pointer to an object to replace the one managed by this CAutoPtr (see page 68). NULL by default.

Description

Resets the object managed by this CAutoPtr (see page 68) to the one specified in pObject. If an object is already managed by this CAutoPtr (see page 68), it is released before assigning the new one. If pObject is NULL, this CAutoPtr (see page 68) no longer manages any object.

2.4.6.2.4 - Operators

2.4.6.2.4.1 - CAutoPtr::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: The CAutoPtr (see page 68) objects manage different object instances.
- false: Both CAutoPtr (see page 68) objects manage the same object instance.

Description

Compares the CAutoPtr (see page 68) on the right hand side of the operator to the CAutoPtr (see page 68) on the left hand side of the operator to verify that they manage different object instances.

See Also

operator=, operator==

2.4.6.2.4.2 - CAutoPtr::* Operator

Reference operator.

C++

```
_Type& operator *() const;
```

Returns

- A reference to the object managed by this CAutoPtr (see page 68).

Description

Returns a reference to the object managed by this object. This allows the CAutoPtr (see page 68) object to be used as if it was the object.

Notes

The managed object must not be NULL, otherwise this operator generates an assertion.

See Also

operator->

2.4.6.2.4.3 - CAutoPtr::< Operator

Less than operator.

C++

```
bool operator <(IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: The CAutoPtr (see page 68) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.
- false: The CAutoPtr (see page 68) object on the left hand side of the operator is greater than the one on the right hand side of the operator.

Description

Verifies that the CAutoPtr (see page 68) on the left hand side of the operator is smaller than the CAutoPtr (see page 68) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator>, operator>=, operator<=

2.4.6.2.4.4 - CAutoPtr::<= Operator

Less than or equal operator.

C++

```
bool operator <=(IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: The CAutoPtr (see page 68) object on the left hand side of the operator is smaller than or equal to the one on the right hand side of the operator.
- false: The CAutoPtr (see page 68) object on the left hand side of the operator is greater than the one on the right hand side of the operator.

Description

Verifies that the CAutoPtr (see page 68) on the left hand side of the operator is smaller than or equal to the CAutoPtr (see page 68) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator>, operator<, operator>=

2.4.6.2.4.5 - =

2.4.6.2.4.5.1 - CAutoPtr::= Operator

Assignment operator.

C++

```
CAutoPtr<_Type>& operator =(IN TOA CAutoPtr<_Type>& rRhs);
```

Parameters

Parameters	Description
IN TOA CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

A reference to the assigned CAutoPtr (see page 68) object.

Description

Assigns the object managed by the CAutoPtr (see page 68) on the right hand side of the operator to the CAutoPtr (see page 68) on the left hand side of the operator. The object managed by the CAutoPtr (see page 68) on the left hand side of the operator is deleted and replaced by the object managed by the CAutoPtr (see page 68) on the right hand side of the operator. The CAutoPtr (see page 68) on the left hand side of the operator stops managing the object, meaning that it no longer has a pointer to the object.

See Also

operator==, operator!=

2.4.6.2.4.5.2 - CAutoPtr::= Operator

Assignment operator.

C++

```
CAutoPtr<_Type>& operator = (IN TOA _Type* pObject);
```

Parameters

Parameters	Description
IN TOA _Type* pObject	Pointer to an object to be managed by this CAutoPtr (see page 68).

Returns

A reference to the assigned CAutoPtr (see page 68) object.

Description

Assigns the object on the right hand side of the operator to the CAutoPtr (see page 68) on the left hand side of the operator. The object managed by the CAutoPtr (see page 68) on the left hand side of the operator is deleted and replaced by the object on the right hand side of the operator.

2.4.6.2.4.6 - CAutoPtr::== Operator

Comparison operator.

C++

```
bool operator == (IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: Both CAutoPtr (see page 68) objects manage the same object instance.
- false: The CAutoPtr (see page 68) objects manage different object instances.

Description

Compares the CAutoPtr (see page 68) on the right hand side of the operator to the CAutoPtr (see page 68) on the left hand side of the operator to verify that they manage the same object instance.

See Also

operator=, operator!=

2.4.6.2.4.7 - CAutoPtr::> Operator

Greater than operator.

C++

```
bool operator > (IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: The CAutoPtr (see page 68) object on the left hand side of the operator is greater than the one on the right hand side of the operator.
- false: The CAutoPtr (see page 68) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.

Description

Verifies that the CAutoPtr (see page 68) on the left hand side of the operator is greater than the CAutoPtr (see page 68) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator<, operator>=, operator<=

2.4.6.2.4.8 - CAutoPtr::-> Operator

Dereference operator.

C++

```
_Type* operator ->() const;
```

Returns

- A pointer to the object managed by this CAutoPtr (see page 68).

Description

Returns a pointer to the object managed by this object. This allows the CAutoPtr (see page 68) object to be used as if it was the object.

Notes

The managed object must not be NULL, otherwise this operator generates an assertion.

See Also

operator*

2.4.6.2.4.9 - CAutoPtr::>= Operator

Greater than or equal operator.

C++

```
bool operator >=(IN const CAutoPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CAutoPtr<_Type>& rRhs	The CAutoPtr (see page 68) object that is on the right hand side of the operator.

Returns

- true: The CAutoPtr (see page 68) object on the left hand side of the operator is greater than or equal to the one on the right hand side of the operator.
- false: The CAutoPtr (see page 68) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.

Description

Verifies that the CAutoPtr (see page 68) on the left hand side of the operator is greater than or equal to the CAutoPtr (see page 68) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator>, operator<, operator<=

2.4.6.3 - CSharedPtr Template

Class that automatically manages an object's lifetime.

Class Hierarchy

CSharedPtr

C++

```
template <class _Type>
class CSharedPtr;
```

Description

The CSharedPtr class automatically manages an object's lifetime. It takes care of managing the object's reference count. The reference to the managed object is automatically released when the object is destroyed. This allows code that uses an object to not worry about the object's lifetime and reference count.

For the CSharedPtr class to work, the object stored MUST implement the AddRef() and ReleaseRef() methods. When created, the object's reference count MUST be 1. Calling AddRef adds a reference on the object, and ReleaseRef() removes one. When the reference count reaches 0, the object MUST delete itself. Concurrency protection, if needed, is left to the implementation of these two methods.

The CSharedPtr class is especially useful when declared on the stack or as a class member since it automatically releases the object once it is destroyed.

Notes

The AddRef() and ReleaseRef() methods MUST NOT be called on the object managed by a CSharedPtr. Doing so would cause the reference count on the object to become invalid, resulting in either early deletion or memory leaks.

Location

Basic/CSharedPtr.h

Example

In the code below, an ECOM (see page 412) interface is acquired via the CreateEComInstance (see page 420) method. That reference to the interface is automatically put in the CSharedPtr object that manages its lifetime. The interface is automatically released once the CSharedPtr object is deleted. In this example, this occurs when the function returns.

```
void Function()
{
    CSharedPtr<IInterface> spInterface;

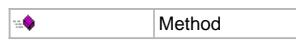
    mxt_result res = CreateEComInstance(CLSID_CInterface, NULL, OUT spInterface);

    if (MX_RIS_S(res))
    {
        spInterface->InterfaceMethod();
    }
}
```

Constructors

Constructor	Description
~CSharedPtr (see page 76)	Default constructor.

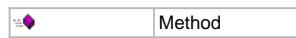
Legend



Destructors

Destructor	Description
~CSharedPtr (see page 77)	Destructor.

Legend



Operators

Operator	Description
!= (see page 77)	Different than operator.
*	Reference operator.

~_Type** (see page 79)	Cast operator to _Type**.
< (see page 79)	Less than operator.
<= (see page 80)	Less than or equal to operator.
= (see page 81)	Assignment operator.
== (see page 81)	Comparison operator.
> (see page 82)	Greater than operator.
>= (see page 83)	Dereference operator.
>= (see page 83)	Greater than or equal to operator.

Legend

	Method
---	--------

Methods

Method	Description
Get (see page 77)	Gets the pointer to the managed object.
Reset (see page 77)	Resets the managed object.

Legend

	Method
---	--------

2.4.6.3.1 - Constructors

2.4.6.3.1.1 - CSharedPtr

2.4.6.3.1.1.1 - CSharedPtr::CSharedPtr Constructor

Default constructor.

C++

```
explicit CSharedPtr(IN _Type* pObject = NULL);
```

Parameters

Parameters	Description
IN _Type* pObject = NULL	Pointer to an object to be managed by this CSharedPtr. NULL by default.

Description

This is the default constructor of the CSharedPtr object. If pObject is not NULL, this object takes its ownership.

See Also

[~CSharedPtr](#)

Example

This example shows different ways to create a CSharedPtr object using its default constructor.

```
CSharedPtr<IInterface> autoInterface;
IInterface* pInterface = ...
CSharedPtr<IInterface> autoInterface(IN pInterface);
pInterface = NULL;
```

2.4.6.3.1.1.2 - CSharedPtr::CSharedPtr Constructor

Copy constructor.

C++

```
CSharedPtr(IN const CSharedPtr<_Type>& rSrc);
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rSrc	A reference to the CSharedPtr object to copy.

Description

This is the copy constructor of the CSharedPtr. It creates a new CSharedPtr object that manages a new reference of the object.

See Also

`~CSharedPtr`

2.4.6.3.2 - **Destructors****2.4.6.3.2.1 - **CSharedPtr::~CSharedPtr Destructor****

Destructor.

C++

```
~CSharedPtr();
```

Description

This is the destructor of the `CSharedPtr` (see page 75) object. It releases the reference on the object it manages.

See Also

`CSharedPtr` (see page 75)

2.4.6.3.3 - **Methods****2.4.6.3.3.1 - **CSharedPtr::Get Method****

Gets the pointer to the managed object.

C++

```
_Type* Get() const;
```

Returns

- A pointer to the object managed by this `CSharedPtr` (see page 75).

Description

Returns a pointer to the object managed by this object. No reference is added to the returned pointer.

2.4.6.3.3.2 - **CSharedPtr::Reset Method**

Resets the managed object.

C++

```
void Reset(IN _Type* pObject = NULL);
```

Parameters

Parameters	Description
<code>IN _Type* pObject = NULL</code>	A pointer to an object to replace the one managed by this <code>CSharedPtr</code> (see page 75). <code>NULL</code> by default.

Description

Resets the object managed by this `CSharedPtr` (see page 75) to the one specified in `pObject`. If an object is already managed by this `CSharedPtr` (see page 75), it is released before assigning the new one. If `pObject` is `NULL`, this `CSharedPtr` (see page 75) no longer manages any object.

2.4.6.3.4 - **Operators****2.4.6.3.4.1 - **!=******2.4.6.3.4.1.1 - **CSharedPtr::!= Operator****

Different than operator.

C++

```
bool operator !=(IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) objects manage different object instances.
- false: Both CSharedPtr (see page 75) objects manage the same object instance.

Description

Compares the CSharedPtr (see page 75) on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator to verify that they manage different object instances.

See Also

operator=, operator==

2.4.6.3.4.1.2 - CSharedPtr::!= Operator

Different than operator.

C++

```
bool operator !=(IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) and the pointer manage different object instances.
- false: The CSharedPtr (see page 75) and the pointer manage the same object instance.

Description

Compares the pointer on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator to verify that they manage different object instances.

See Also

operator=, operator==

2.4.6.3.4.2 - CSharedPtr::* Operator

Reference operator.

C++

```
_Type& operator *() const;
```

Returns

- A reference to the object managed by this CSharedPtr (see page 75).

Description

Returns a reference to the object managed by this object. This allows the CSharedPtr (see page 75) object to be used as if it was the object.

Notes

The object managed must be non-NULL, otherwise this operator generates an assertion.

See Also

operator->

2.4.6.3.4.3 - CSharedPtr::_Type** Operator

Cast operator to `_Type**`.

C++

```
operator _Type**();
```

Returns

- The address to the pointer to the object managed by this `CSharedPtr` (see page 75).

Description

Returns the address to the pointer to the object managed by this object. This allows the `CSharedPtr` (see page 75) object to be used when a function takes the address of a pointer to the object as an out parameter.

```
void Get(OUT IInterface** ppInterface)
{
    *ppInterface = ...
}

void Function()
{
    CSharedPtr<IInterface> spInterface;

    mxt_result res = Get(OUT spInterface);

    if (MX_RIS_S(res))
    {
        spInterface->InterfaceMethod();
    }
}
```

2.4.6.3.4.4 - <

2.4.6.3.4.4.1 - CSharedPtr::< Operator

Less than operator.

C++

```
bool operator <(IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The <code>CSharedPtr</code> (see page 75) object that is on the right hand side of the operator.

Returns

- true: The `CSharedPtr` (see page 75) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.
- false: The `CSharedPtr` (see page 75) object on the left hand side of the operator is greater than the one on the right hand side of the operator.

Description

Verifies that the `CSharedPtr` (see page 75) on the left hand side of the operator is smaller than the `CSharedPtr` (see page 75) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

`operator>`, `operator>=`, `operator<=`

2.4.6.3.4.4.2 - CSharedPtr::< Operator

Less than operator.

C++

```
bool operator <(IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than the pointer on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than the pointer on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is smaller than the pointer on the right hand side of the operator. The comparison is done using the value of the pointer managed by the CSharedPtr (see page 75) object and pObject.

See Also

operator>, operator>=, operator<=

2.4.6.3.4.5 - <=

2.4.6.3.4.5.1 - CSharedPtr::<= Operator

Less than or equal to operator.

C++

```
bool operator <=( IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than or equal to the one on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than the one on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is smaller or equal than the CSharedPtr (see page 75) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator>, operator<, operator>=

2.4.6.3.4.5.2 - CSharedPtr::<= Operator

Less than or equal to operator.

C++

```
bool operator <=( IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than or equal to the pointer on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than the pointer on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is smaller than or equal to the pointer on the right hand side of the operator. The comparison is done using the value of the pointer managed by the CSharedPtr (see page 75) object and pObject.

See Also

operator>, operator<, operator>=

2.4.6.3.4.6 - =

2.4.6.3.4.6.1 - CSharedPtr::= Operator

Assignment operator.

C++

```
CSharedPtr<_Type>& operator =(IN _Type* pObject);
```

Parameters

Parameters	Description
IN _Type* pObject	Pointer to an object to be managed by this CSharedPtr (see page 75).

Returns

A reference to the assigned CSharedPtr (see page 75) object.

Description

Assigns the object on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator.

2.4.6.3.4.6.2 - CSharedPtr::= Operator

Assignment operator.

C++

```
CSharedPtr<_Type>& operator =(IN const CSharedPtr<_Type>& rRhs);
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

A reference to the assigned CSharedPtr (see page 75) object.

Description

Assigns the object managed by the CSharedPtr (see page 75) on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator. The reference held by the CSharedPtr (see page 75) on the right hand side of the operator is released.

See Also

operator==, operator!=

2.4.6.3.4.7 - ==

2.4.6.3.4.7.1 - CSharedPtr::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

- true: Both CSharedPtr (see page 75) objects manage the same object instance.
- false: The CSharedPtr (see page 75) objects manage different object instances.

Description

Compares the CSharedPtr (see page 75) on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator to verify that they manage the same object instance.

See Also

operator=, operator!=

2.4.6.3.4.7.2 - CSharedPtr::== Operator

Comparison operator.

C++

```
bool operator ==(IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) and the pointer manage the same object instance.
- false: The CSharedPtr (see page 75) and the pointer manage different object instances.

Description

Compares the pointer on the right hand side of the operator to the CSharedPtr (see page 75) on the left hand side of the operator to verify that they manage the same object instance.

See Also

operator=, operator!=

2.4.6.3.4.8 - >

2.4.6.3.4.8.1 - CSharedPtr::> Operator

Greater than operator.

C++

```
bool operator >(IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than the one on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is greater than the CSharedPtr (see page 75) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator<, operator>=, operator<=

2.4.6.3.4.8.2 - CSharedPtr::> Operator

Greater than operator.

C++

```
bool operator >(IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than the pointer on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than the pointer on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is greater than the pointer on the right hand side of the operator. The comparison is done using the value of the pointer managed by the CSharedPtr (see page 75) object and pObject.

See Also

operator<, operator>=, operator<=

2.4.6.3.4.9 - CSharedPtr::-> Operator

Dereference operator.

C++

```
_Type* operator ->() const;
```

Returns

- A pointer to the object managed by this CSharedPtr (see page 75).

Description

Returns a pointer to the object managed by this object. This allows the CSharedPtr (see page 75) object to be used as if it was the object.

Notes

The object managed must be non-NULL, otherwise this operator generates an assertion.

See Also

operator*

2.4.6.3.4.10 - >=**2.4.6.3.4.10.1 - CSharedPtr::>= Operator**

Greater than or equal to operator.

C++

```
bool operator >=(IN const CSharedPtr<_Type>& rRhs) const;
```

Parameters

Parameters	Description
IN const CSharedPtr<_Type>& rRhs	The CSharedPtr (see page 75) object that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than or equal to the one on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than the one on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is greater than or equal to the CSharedPtr (see page 75) on the right hand side of the operator. The comparison is done using the value of the pointers to the managed objects.

See Also

operator>, operator<, operator<=

2.4.6.3.4.10.2 - CSharedPtr::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const _Type* pObject) const;
```

Parameters

Parameters	Description
IN const _Type* pObject	The pointer that is on the right hand side of the operator.

Returns

- true: The CSharedPtr (see page 75) object on the left hand side of the operator is greater than or equal to the pointer on the right hand side of the operator.
- false: The CSharedPtr (see page 75) object on the left hand side of the operator is smaller than the pointer on the right hand side of the operator.

Description

Verifies that the CSharedPtr (see page 75) on the left hand side of the operator is greater than or equal to the pointer on the right hand side of the operator. The comparison is done using the value of the pointer managed by the CSharedPtr (see page 75) object and pObject.

See Also

operator>, operator<, operator<=

2.4.7 - Types

This section documents the types of the Sources/Basic folder.

Types

Type	Description
int16_t (see page 85)	Standard data type used throughout M5T's software.
int32_t (see page 85)	Standard data type used throughout M5T's software.
int64_t (see page 85)	Standard data type used throughout M5T's software.
int8_t (see page 85)	Standard data type used throughout M5T's software.
mxt_opaque (see page 87)	Custom data type that allows opaque type usage.
mxt_PFNAssertCallStackTraceHandler (see page 87)	Defines the prototype for a call stack trace handler.
mxt_PFNAssertFailHandler (see page 88)	Defines the prototype for an assertion failed handler.
mxt_PFNAssertFinalBehaviorHandler (see page 88)	Defines the prototype for a final behaviour handler.
mxt_PFNAssertTraceHandler (see page 88)	Defines the prototype for a trace handler.
mxt_pfnEventObserver (see page 89)	Definition of the callback function format.
mxt_PFNTraceCallStackHandler (see page 89)	Defines the prototype for a call stack trace handler.
mxt_PFNTraceFormatHandler (see page 89)	Defines the prototype for a trace format handler.
mxt_PFNTraceGenericFormatHandler (see page 91)	Defines the prototype for a generic format handler.
mxt_PFNTraceOutputHandler (see page 92)	Defines the prototype for a trace output handler.
mxt_result (see page 92)	Data type used to store result information.
mxt_UNALIGNED_INT16 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.

mxt_UNALIGNED_INT32 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.
mxt_UNALIGNED_INT64 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.
mxt_UNALIGNED_UINT16 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.
mxt_UNALIGNED_UINT32 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.
mxt_UNALIGNED_UINT64 (see page 93)	M5T Framework defines data types for platforms not supporting unaligned access.
uint16_t (see page 85)	Standard data type used throughout M5T's software.
uint32_t (see page 85)	Standard data type used throughout M5T's software.
uint64_t (see page 85)	Standard data type used throughout M5T's software.
uint8_t (see page 85)	Standard data type used throughout M5T's software.

2.4.7.1 - Standard data types

Standard data type used throughout M5T's software.

C++

```
typedef signed char int8_t;
typedef unsigned char uint8_t;
typedef signed short int16_t;
typedef unsigned short uint16_t;
typedef signed int int32_t;
typedef unsigned int uint32_t;
typedef long long int int64_t;
typedef unsigned long long int uint64_t;
```

Description

M5T's software is moving towards the adoption of both the standard ANSI C/C++ types and the more recent ISO C99 Exact Integral Types.

If the ISO C99 types are not supported or defined by your compiler, then the file Basic/MxDef.h defines these types to properly compile M5T's software.

For more information about how these types are defined for your specific compiler and platform, please refer to the up-to-date definition found in Basic/MxDef.h.

Standard ANSI C/C++ Types

The following standard types are used:

bool	[true or false]
char	[System dependent]
int	[System dependent]
unsigned int	[System dependent]
float	[±1.175494351 E - 38 .. ±3.402823466 E + 38]
double	[±2.2250738585072014 E - 308 .. ±1.7976931348623158 E + 308]
void*	[Pointer to any variable type]

ISO C99 Exact Integral Types

ISO C99 Exact Integral Types defines the following types:

int8_t	[-128 .. 127]
int16_t	[-32 768 .. 32 767]
int32_t	[-2 147 483 648 .. 2 147 483 647]
int64_t	[-9 223 372 036 854 775 806 .. 9 223 372 036 854 775 807]
uint8_t	[0 .. 255]

uint16_t	[0 .. 65 535]
uint32_t	[0 .. 4 294 967 295]
uint64_t	[0 .. 18 446 744 073 709 551 615]

Deprecated Data Types

The following data types are still used within some of the M5T's software but it is being slowly replaced by the previously defined data types.

CHAR	(Use "char" instead)
UCHAR	(Use "uint8_t" instead)
SCHAR	(Use "int8_t" instead)
SHORT	(Use "int16_t" instead)
USHORT	(Use "uint16_t" instead)
SSHORT	(Use "int16_t" instead)
LONG	(Use "long" instead)
ULONG	(Use "unsigned long" instead)
SLONG	(Use "long" instead)
INT	(Use "int" instead)
UINT	(Use "unsigned int" instead)
SINT	(Use "int" instead)
INT8	(Use "int8_t" instead)
UINT8	(Use "uint8_t" instead)
SINT8	(Use "int8_t" instead)
INT16	(Use "int16_t" instead)
UINT16	(Use "uint16_t" instead)
SINT16	(Use "int16_t" instead)
INT32	(Use "int32_t" instead)
UINT32	(Use "uint32_t" instead)
SINT32	(Use "int32_t" instead)
INT64	(Use "int64_t" instead)
UINT64	(Use "uint64_t" instead)
SINT64	(Use "int64_t" instead)
FLOAT	(Use "float" instead)
DOUBLE	(Use "double" instead)

Location

Basic/MxDef.h

2.4.7.2 - mxt_opaque Type

Custom data type that allows opaque type usage.

C++

```
typedef void* mxt_opaque;
```

Description

The mxt_opaque custom data type allows opaque type usage. It is defined as a void* to be able to contain a pointer regardless of the architecture (32 or 64 bits). Since a mxt_opaque is a pointer, it is necessary to use a reinterpret_cast to assign a non pointer value to a mxt_opaque. To ease the mxt_opaque conversions, the following macros are available:

- MX_VOIDPTR_TO_OPQ (see page 59)(pParam)
- MX_OPQ_TO_VOIDPTR (see page 59)(opq)
- MX_INT32_TO_OPQ (see page 60)(nParam)
- MX_OPQ_TO_INT32 (see page 60)(opq)
- MX_UINT32_TO_OPQ (see page 61)(uParam)
- MX_OPQ_TO_UINT32 (see page 61)(opq)

For example, use the following code to assign an integer to a mxt_opaque:

```
mxt_opaque opq = MX_INT32_TO_OPQ(1);
```

Since it is not permitted to do arithmetics on a void*, it is not permitted on a mxt_opaque either. For example, the following code is not valid:

```
mxt_opaque opq = MX_INT32_TO_OPQ(0);
opq++;
```

One should use a variation of the following code instead:

```
mxt_opaque opq = MX_UINT32_TO_OPQ(0);
unsigned int uValue = 0;
uValue++;
opq = MX_UINT32_TO_OPQ(uValue);
```

When assigning a class member that is a pointer to a mxt_opaque from within a const method, it is necessary to do a const cast followed by a reinterpret_cast, as in the following code:

```
class CExample
{
public:
    void GetOpaqueCount(OUT mxt_opaque opq) const
    {
        opq = MX_VOIDPTR_TO_OPQ(const_cast<unsigned int*>(&m_uCount));
    };
private:
    unsigned int m_uCount;
};
```

Location

Basic/MxDef.h

2.4.7.3 - mxt_PFNAssertCallStackTraceHandler Type

Defines the prototype for a call stack trace handler.

C++

```
typedef void (* mxt_PFNAssertCallStackTraceHandler)(IN mxt_opaque opq);
```

Parameters

Parameters	Description
opq	The opaque value provided at registration.

Description

This data type defines a function prototype for a call stack trace handler that is called by the default mxt_PFNAssertFailHandler (see page 88) when an assertion fails.

2.4.7.4 - mxt_PFNAssertFailHandler Type

Defines the prototype for an assertion failed handler.

C++

```
typedef void (* mxt_PFNAssertFailHandler)(IN mxt_opaque opq, IN const char* pszAssertion, IN int nErrNumber, IN const char* pszExtraComment, IN const char* pszFile, IN unsigned int uLine);
```

Parameters

Parameters	Description
opq	The opaque value provided at registration.
pszAssertion	String describing the assertion.
nErrNumber	When available, the error number associated with the assertion failure.
pszExtraComment	Additional information that can be passed to the user.
pszFile	File in which the assertion failed.
uLine	Line number where the assertion failed.

Description

This data type defines a function prototype for the assertion failed handler, which is responsible for handling failed assertions and notifications most of the time, by using other handlers like mxt_PFNAssertTraceHandler (see page 88).

2.4.7.5 - mxt_PFNAssertFinalBehaviorHandler Type

Defines the prototype for a final behaviour handler.

C++

```
typedef void (* mxt_PFNAssertFinalBehaviorHandler)(IN mxt_opaque opq);
```

Parameters

Parameters	Description
opq	The opaque value provided at registration.

Description

This data type defines a function prototype for a final behaviour handler called after assertion failure.

When an assertion fails and the final behaviour handler is enabled and MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL (see page 273) is defined, the default behaviour is to call `exit()`.

See Also

`MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL` (see page 273), `MXD_ASSERT_FINAL_BEHAVIOR_DISABLE` (see page 273), `MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE` (see page 273)

2.4.7.6 - mxt_PFNAssertTraceHandler Type

Defines the prototype for a trace handler.

C++

```
typedef void (* mxt_PFNAssertTraceHandler)(IN mxt_opaque opq, IN const char* pszAssertion, IN int nErrNumber, IN const char* pszExtraComment, IN const char* pszFile, IN unsigned int uLine);
```

Parameters

Parameters	Description
opq	The opaque value provided at registration.
pszAssertion	String describing the assertion.
nErrNumber	When available, the error number associated with the assertion failure.
pszExtraComment	Additional information that can be passed to the user.
pszFile	File in which the assertion failed.
uLine	Line number where the assertion failed.

Description

This data type defines a function prototype for tracing assertion failed information. It is called by the default mxt_PFNAssertFailHandler (see page 88) when an assertion fails.

2.4.7.7 - mxt_pfnEventObserver Type

Definition of the callback function format.

C++

```
typedef void (* mxt_pfnEventObserver)(IN void* pContext, IN EEventNotifier eEvent, IN va_list args);
```

Parameters

Parameters	Description
pContext	The context associated with the observer.
eEvent	The event to observe.
args	Arguments passed to the observer, depending on the event being observed.

Description

Definition of the callback function format.

2.4.7.8 - mxt_PFNTraceCallStackHandler Type

Defines the prototype for a call stack trace handler.

C++

```
typedef void (* mxt_PFNTraceCallStackHandler)(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN void *p1);
```

Parameters

Parameters	Description
eLevel	The trace's tracing level (may be a combination of one or more levels), from the enumeration EMxTraceLevel (see page 16).
uTraceUniqueId	The trace's unique ID.
p1	Pointer to contextual data.

Description

This data type defines a function prototype for a call stack handler, which is responsible to output a set of traces that shows the contents of the call stack.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.7.9 - mxt_PFNTraceFormatHandler Type

Defines the prototype for a trace format handler.

C++

```
typedef int (* mxt_PFNTraceFormatHandler)(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, INOUT char* pszTraceData, IN const char* pszMsgFormat, IN va_list args, IN unsigned int uTraceSequenceNumber);
```

Parameters

Parameters	Description
eLevel	The trace's tracing level from the enumeration EMxTraceLevel (see page 16).
uTraceUniqueId	The trace's unique Id.
pszTraceData	A pointer to the buffer to be used for tracing.
pszMsgFormat	A pointer to a null-terminated character string that contains the 'printf' format of the user's trace. This parameter should NOT be NULL.
args	The variable list of arguments associated with the format in pszMsgFormat.
uTraceSequenceNumber	The sequence number for the trace that is to be formatted. The sequence number allows to detect missing traces.

Returns

Size of the data actually written in the buffer.

Description

This data type defines a function prototype for a trace format handler, which is responsible for building the actual trace to output by mxt_PFNTraceOutputHandler (see page 92). It is recommended that all trace format handlers check if pszTraceData is not NULL before attempting to use it.

Default format handler

The default format handler prepares traces with the following format:

```
<01> <02> <03> | <04> | <05> | <06> | <07> | <08> | <09> | <10>
```

The numeric values are printed in decimal (base 10), except if otherwise noted.

The separator "|" is MANDATORY even if a field is disabled. It allows to keep track of what is being traced.

The two digit number inside the <00> refers to the position of a given field inside the entire trace.

The default trace format handler is adapted to support the SYSLOG format. The SYSLOG format requires that a trace is made up of a header followed by trace data. Therefore, with respect to the SYSLOG format, the fields 01, 02, and 03 make up the SYSLOG header while the following "|<04>...<11>" consists in the SYSLOG trace data. Again, the space (' ') is used as separator between the fields 01-02 and 02-03 to follow the SYSLOG format.

<01> - Syslog PRI

The Syslog PRI is calculated according to RFC 3164: Facility * 8 + Severity. The Facility is always user-level messages (1). The syslog severity is calculated from the trace level (EMxTraceLevel (see page 16)). For example, from eLEVEL6, this yields PRI <14>, since (Facility * 8) + Severity = (1 * 8) + 6 = 14. The following is a list of all the possible PRIs for all trace levels:

- eLEVEL0 -> <8>,
- eLEVEL1 -> <9>,
- eLEVEL2 -> <10>,
- eLEVEL3 -> <11>,
- eLEVEL4 -> <12>,
- eLEVEL5 -> <13>,
- eLEVEL6 -> <14>,
- eLEVEL7 -> <15>,
- eLEVEL8 -> <15>,
- eLEVEL9 -> <15>.

This field is enabled by default.

<02> - Local Date and Time

Local Time and Date in Syslog format ("Mmm dd hh:mm:ss"). This field is not enabled by default.

<03> - Hostname

The local hostname or IP address. This field is not enabled by default. See how to enable/disable it below with eMXTFIELD_HOSTNAME.

<04> - Tracing level

The tracing level used to output the trace. This field is enabled by default See how to enable/disable it below with eMXTFIELD_TRACINGLEVEL.

<05> - Program Name

An optional global program name that can be defined using the preprocessor macro MXD_TRACE_PROGNAME (see page 312), exactly as <11>. This field is not enabled by default.

<06> - Process ID

The process ID, in hexadecimal. Note that it is not available on every operating system and yields 0 in these cases. This field is not enabled by default.

<07> - Thread ID

The thread ID, in hexadecimal. Note that it is not available on every operating system and yields 0 in these cases. This field is enabled by default.

<08> - System Timestamp

The time stamp. It is a 64-bits integer that gives the time the system is up, in milliseconds. This field is enabled by default. See how to enable/disable it below with `eMXTFIELD_TIME_STAMP`.

<09> - Sequence Number

The sequence number. This number grows by one for each trace, and so each trace has a different number. This field is enabled by default.

<10> - Actual Trace

The trace, with a newline appended. This field is always present.

All the field can be enabled or disabled using the following functions:

- `MxTraceEnableField` (see page 26)
- `MxTraceDisableField` (see page 24)
- `MxTraceEnableDefaultFields` (see page 26)
- `MxTraceEnableSyslogFields` (see page 28)
- `MxTraceEnableAllFields` (see page 25)
- `MxTraceDisableAllFields` (see page 24)
- `MxTraceFieldIsEnabled` (see page 29)

The hostname field can be changed with the following function: `MxTraceSetHostNameField` (see page 31)

Location

Defined in `Basic/MxTrace.h` but must include `Config/MxConfig.h` to access this.

See Also

General Tracing Configuration (see page 2)

2.4.7.10 - `mxt_PFNTraceGenericFormatHandler` Type

Defines the prototype for a generic format handler.

C++

```
typedef int (* mxt_PFNTraceGenericFormatHandler)(INOUT char* pszBuffer, IN int nBufferSize);
```

Parameters

Parameters	Description
<code>pszBuffer</code>	The buffer where to write the formatted text.
<code>nBufferSize</code>	The maximum number of bytes available for the formatted text.

Returns

The number of bytes actually written to the buffer, excluding the null-terminator.

Description

This data type defines a function prototype for a generic format handler. For instance, this format handler can be used to format time, in which case it is responsible for getting and formatting the actual time stamp that is set in the trace itself.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.7.11 - mxt_PFNTraceOutputHandler Type

Defines the prototype for a trace output handler.

C++

```
typedef void (* mxt_PFNTraceOutputHandler)(IN EMxTraceLevel eLevel, IN uint32_t uTraceUniqueId, IN const char* pszTrace, IN int nMsgSize);
```

Parameters

Parameters	Description
eLevel	The trace's tracing level.
uTraceUniqueId	The trace's unique ID.
pszTrace	A pointer to a null-terminated character string that contains the whole trace with a newline.
nMsgSize	The trace length, excluding the null-terminator.

Description

This data type defines a function prototype for a trace output handler, which is responsible for outputting the trace.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.4.7.12 - mxt_result Type

Data type used to store result information.

C++

```
typedef int32_t mxt_result;
```

Description

The `mxt_result` data type, which is defined on a signed integer, is used as a standard mechanism for methods to return a result or error. There are three distinct usable fields in a `mxt_result`: The level, the package ID, and the result code ID. A field has been reserved for future use. See below for the definition of those fields.

The result mechanism is defined to add very little overhead while returning a lot of information. There is also a list of generic error messages that are not associated directly with a package, therefore avoiding redundant messages.

Information messages can also be associated with every error message in case there is the need to output readable information. See [MX_RGET_MSG_STR](#) (see page 50) for more details.

The result mechanism is also designed so it can be used in a user application. To do so, the package ID enumeration must be augmented. New error codes can also be created to fit the application needs. See `MXD_PKG_ID_OVERRIDE` (see page 300) and `MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN` (see page 49).

mxt result definition:

The mxt result is defined as one signed 32 bits integer.

```

MSB                                     LSB
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+
| LEV | Reserved |     Package ID     |           Code ID  |
+-----+-----+-----+-----+

```

Bits

31-30: Level

Bits 31 30 definition:

- 0 0 - SUCCESS-INFORMATION
- 0 1 - SUCCESS-WARNING

```

1 0 - FAIL-ERROR
1 1 - FAIL-CRITICAL

SUCCESS-INFORMATION: Operation Succeeded and Information may be
                     returned.
                     i.e.: resS_OK (CodeID = 0, no info)
                           resSI_TRUE (CodeID > 0, info = TRUE)
                           resSI_FALSE (CodeID > 0, info = FALSE)

SUCCESS-WARNING:      Operation Succeeded and a Warning is returned.
                     i.e.: resSW_MEMORY_LOW

FAIL-ERROR:           Operation Failed and the Error cause is
                     returned.
                     i.e.: resFE_INVALID_ARGUMENT

FAIL_CRITICAL:        Operation Failed and a Critical error cause is
                     returned.
                     i.e.: resFC_DATA_CORRUPTION

29-24:   Reserved
          Reserved for future use (Shall always be 0)

23-14:   Package ID (10 bits, up to 1024 packages)
          Package ID is unique for each package.

13-0:    Code ID (14 bits, up to 16384 codes for each level)
          Codes between 0 and MX_RESULT_NB_MAX_SHARED_CODE_ID - 1 ( 0-1023 )
          are reserved for shared code IDs.
          This range of codes are not redefined in each package. Instead, they
          are defined only once and can be used in any packages. Code ID codes
          starting from MX_RESULT_NB_MAX_SHARED_CODE_ID can be defined for
          each package and each level.

```

Result code IDs:

Result code IDs shall be defined in an enum as described below:

resLEVEL_PKGNAME_MSG

res: (lower case) mxt_result standard prefix.

LEVEL: (upper case) SI_ for Success Information (note that S_OK is an exception) SW_ for Success Warning FE_ for Fail Error FC_ for Fail Critical

PKGNAME: (upper case) Package name, with regards to EMxPackageld (see page 12) nomenclature (omitted for shared code ID).

MSG: (upper case) Result message. Each word shall be separated with an underscore.

Code ID = 0 is reserved for resS_OK. This allows a faster detection of a SUCCESS without information. A result of that constraint is that other shared code IDs must start with code ID = 1 instead of code ID = 0. Note that shared messages table must also start with an empty msg (matches code ID 0 which is not used) to make code ID 1 match with msg 1 and so on.

Notes

Since declaration result codes for each package are based on an enum, the enum type must be int32_t (see page 85).

Location

Defined in Basic/Result.h but must include Config/MxConfig.h to access this.

See Also

MX_RGET_PKG_BASE_SUCCESS_WARNING_CODE_ID (see page 49), MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN (see page 49), MX_RIS_S (see page 50)

2.4.7.13 - Unaligned data types

M5T Framework defines data types for platforms not supporting unaligned access.

C++

```

typedef SUalignedInt16 mxt_UNALIGNED_INT16;
typedef SUalignedUInt16 mxt_UNALIGNED_UINT16;
typedef SUalignedInt32 mxt_UNALIGNED_INT32;
typedef SUalignedUInt32 mxt_UNALIGNED_UINT32;
typedef SUalignedInt64 mxt_UNALIGNED_INT64;
typedef SUalignedUInt64 mxt_UNALIGNED_UINT64;

```

Description

Special type definitions for self-alignment on misaligned data access.

To avoid misalignment problems, these particular types are created for processor architectures that do not handle misalignment exceptions. For example, the MIPS32 core used on some processors.

The following types are defined:

- `mxt_UNALIGNED_INT16` for signed 16 bits access;
- `mxt_UNALIGNED_UINT16` for unsigned 16 bits access;
- `mxt_UNALIGNED_INT32` for signed 32 bits access;
- `mxt_UNALIGNED_UINT32` for unsigned 32 bits access;
- `mxt_UNALIGNED_INT64` for signed 64 bits access;
- `mxt_UNALIGNED_UINT64` for unsigned 64 bits access.

The above types map to the corresponding standard types for platforms that do not require aligned access.

Location

`Basic/MxDefUnalignedType.h`

2.4.8 - Variables

This section documents the variables of the Sources/Basic folder.

2.4.8.1 - `g_szEMPTY_STRING` Variable

Empty c-style string used to replace NULL strings.

C++

```
const char g_szEMPTY_STRING[];
```

Description

This symbol provides access to an empty c-style string that can be used in place of NULL strings.

2.4.8.2 - `g_szNULL` Variable

c-style string indicating NULL strings.

C++

```
const char g_szNULL[];
```

Description

This symbol provides access to a c-style string that denotes a NULL string in the same way on all platforms. It is primarily used in the `MX_MAKE_STRING_NULL_SAFE` (see page 58) macro. It should not be used directly.

See Also

`MX_MAKE_STRING_NULL_SAFE` (see page 58)

2.5 - Cap

This section documents the Sources/Cap folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 94)
- Structures (see page 157)
- Templates (see page 158)

2.5.1 - Classes

This section documents the classes of the Sources/Cap folder.

Classes

Class	Description
CBlob (see page 95)	This class is defined as a Binary Large OBject (Blob).
CBlockAllocator (see page 114)	Manages a list of free blocks.
CMarshaler (see page 117)	Class used to store objects.
CMemoryQueue (see page 121)	Implements a fixed size queue containing memory blocks of variable length.
CString (see page 126)	Class used to encapsulate a c-style string.
CSubAllocator (see page 146)	This class performs sub-allocation in a huge memory block allocated upon creation of the class.
CVersion (see page 149)	Class used for version checking.
IAllocator (see page 156)	Interface allowing the allocation and deallocation of free blocks.

2.5.1.1 - CBlob Class

This class is defined as a Binary Large OBject (Blob).

Class Hierarchy



C++

```
class CBlob : protected CVectorBase;
```

Description

This class is defined as a Binary Large OBject (Blob). The Blob holds an array of bytes and offers some operations to manipulate this buffer safely. The security of the blob can be bypassed for performance purposes. To do so, the client can call GetFirstIndexPtr (see page 102) or GetEndIndexPtr (see page 101) and use the pointer to the array of bytes directly.

The blob inherits all of CVectorBase's capabilities where the size of each element is one byte. The blob totally relies on CVectorBase for all standard vector operations.

The index in the array of bytes is always zero-based.

Location

Cap/CBlob.h

Constructors

Constructor	Description
CBlob (see page 96)	Default constructor.

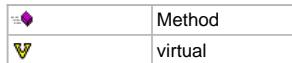
Legend



Destructors

Destructor	Description
~CBlob (see page 97)	Destructor.

Legend



Operators

Operator	Description
!= (see page 111)	Different than operator.
[] (see page 111)	Returns a reference to the byte at ulIndex.
< (see page 112)	Less than operator.
<= (see page 112)	Less than or equal to operator.
= (see page 112)	Assignment operator.
== (see page 113)	Comparison operator.
> (see page 113)	Greater than operator.
>= (see page 113)	Greater than or equal to operator.

Legend

	Method
--	--------

Methods

Method	Description
Append (see page 98)	Appends the content of another blob at the end index.
AppendBits (see page 99)	Appends bits at the end of the blob.
Erase (see page 100)	Erases the byte at ulIndex from the blob.
EraseAll (see page 100)	Erases all the bytes from the blob.
GetAt (see page 100)	Returns a reference to the byte at ulIndex.
GetCapacity (see page 101)	Returns the capacity.
GetEndIndex (see page 101)	Returns the index of the first unused byte.
GetEndIndexPtr (see page 101)	Returns a pointer to the byte at the end index.
GetFirstIndex (see page 102)	Returns the index of the first used byte.
GetFirstIndexPtr (see page 102)	Returns a pointer to the byte at index zero.
GetLastIndex (see page 102)	Returns the index of the last used byte.
GetLockCapacity (see page 103)	Returns the lock capacity count.
GetSize (see page 103)	Gets the size.
GetUnreadBits (see page 103)	Gets the number of unread bits.
GetUnreadSize (see page 103)	Gets the remaining number of bytes to read from the blob.
Insert (see page 104)	Inserts the bytes contained in another blob.
IsEmpty (see page 105)	Checks if the blob is empty.
IsFull (see page 105)	Checks if the blob is full.
LockCapacity (see page 106)	Locks the capacity.
Merge (see page 106)	Moves all the byte of another blob to a specific index.
PadBits (see page 106)	Pads with cleared ('0') bits at the end of the blob.
Read (see page 106)	Reads bytes starting at the current read index.
ReadBits (see page 107)	Reads bits at the current read index.
ReadNoCopy (see page 107)	Reads bytes starting at the current read index.
RealignAppendBits (see page 108)	Pads the needed number of bits at the end of the blob so that it is realigned on a multiple of the given number of bits.
RealignReadBits (see page 108)	Increments (if needed) the read index so that it is a multiple of the given number of bits.
ReduceCapacity (see page 108)	Reduces the capacity.
ReserveCapacity (see page 109)	Increases the capacity.
Resize (see page 109)	Resizes the blob.
SetReadIndex (see page 110)	Sets the current read index.
SkipBits (see page 110)	Skips bits before reading.
Split (see page 110)	Moves some bytes from this blob to another starting from a specific index.
Swap (see page 111)	This method exchanges the position of two bytes.
UnlockCapacity (see page 111)	Unlocks the capacity.

Legend

	Method
--	--------

2.5.1.1.1 - Constructors

2.5.1.1.1.1 - CBlob

2.5.1.1.1.1.1 - CBlob::CBlob Constructor

Default constructor.

C++

```
CBlob(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the internal buffer.

Description

Constructor. With this variant of the constructor, the capacity of the internal buffer is zero.

2.5.1.1.1.1.2 - CBlob::CBlob Constructor

Copy constructor.

C++

```
CBlob(IN const CBlob& rBlob);
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Description

Copy constructor.

2.5.1.1.1.1.3 - CBlob::CBlob Constructor

Constructor. Allocates capacity and inserts data.

C++

```
CBlob(IN const uint8_t* puData, IN unsigned int uDataSize, IN unsigned int uCapacity, IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to the data to be inserted into the blob.
IN unsigned int uDataSize	The size of the data pointed to by puData.
IN unsigned int uCapacity	The initial capacity of the internal buffer.
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the internal buffer.

Description

Constructor. With this variant of the constructor, the capacity of the internal buffer is the maximum value between uDataSize and uCapacity. The blob buffer is filled with uDataSize bytes of the data pointed to by puData.

2.5.1.1.1.1.4 - CBlob::CBlob Constructor

Constructor. Allocates capacity.

C++

```
CBlob(IN unsigned int uCapacity, IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN unsigned int uCapacity	The initial capacity of the internal buffer.
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the internal buffer.

Description

Constructor. With this variant of the constructor, the capacity of the internal buffer is uCapacity.

2.5.1.1.2 - Destructors

2.5.1.1.2.1 - CBlob::~CBlob Destructor

Destructor.

C++

```
virtual ~CBlob();
```

Description

Destructor.

2.5.1.1.3 - Methods

2.5.1.1.3.1 - Append

2.5.1.1.3.1.1 - CBlob::Append Method

Appends the content of another blob at the end index.

C++

```
mxt_result Append(IN const CBlob& rBlob, IN unsigned int uSrcBlobIndex = 0, IN unsigned int uSize = uBLOB_COPY_ALL);
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the blob to be appended.
IN unsigned int uSrcBlobIndex = 0	Index of the first byte to append from rBlob (default is 0).
IN unsigned int uSize = uBLOB_COPY_ALL	Size in bytes to append from rBlob (default is entire blob).

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

This method partially or fully appends the content of rBlob to the current blob. When uSrcBlobIndex and uSize are left to the default value, the content of rBlob is completely appended. When uSrcBlobIndex and uSize are specified, uSize bytes of rBlob starting at uSrcBlobIndex are appended.

See Also

Insert (see page 104)

2.5.1.1.3.1.2 - CBlob::Append Method

Appends bytes at the end index.

C++

```
mxt_result Append(IN const char* pszString);
```

Parameters

Parameters	Description
IN const char* pszString	A pointer to the c-style string buffer to insert.

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

This method appends a c-style string AS BYTES at the end index. The string to be inserted is pointed by pszString. The terminating '0' character is not inserted.

The parameter pszString MUST be null-terminated, otherwise the result is unpredictable. The method uses strlen to get the number of bytes to append.

See Also

Insert (see page 104)

2.5.1.1.3.1.3 - CBlob::Append Method

Appends bytes at the end index.

C++

```
mxt_result Append(IN const uint8_t* puData, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN const uint8_t* puData	A pointer to the bytes to insert.
IN unsigned int uSize	The number of bytes to insert.

Returns

```
resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY
```

Description

This method appends uSize bytes of data at the end index. The bytes to be inserted are pointed by puData.

See Also

[Insert \(see page 104\)](#)

2.5.1.1.3.1.4 - CBlob::Append Method

Appends a byte at the end index.

C++

```
mxt_result Append(IN uint8_t uByte);
```

Parameters

Parameters	Description
IN uint8_t uByte	Byte to append to the buffer.

Returns

```
resS_OK resFE_OUT_OF_MEMORY
```

Description

This method appends a single byte at the end index.

See Also

[Insert \(see page 104\)](#)

2.5.1.1.3.2 - CBlob::AppendBits Method

Appends bits at the end of the blob.

C++

```
void AppendBits(IN const uint8_t* puData, IN unsigned int uSizeInBits, IN unsigned int uStartingBit = 0);
```

Parameters

Parameters	Description
IN const uint8_t* puData	A pointer to the bytes containing the bits to append.
IN unsigned int uSizeInBits	The number of bits to append.
IN unsigned int uStartingBit = 0	The position of the first bit to append. The position should be included in the range 0 - 7 where 0 is the MSB.

Description

This method appends uSizeInBits bits of data at the end of the blob. Successive calls to this method result in packing the bits just after the bits previously appended. Any call to [Insert \(see page 104\)](#) or [Append \(see page 98\)](#) causes a realignment on a one byte boundary.

See Also

[ReadBits \(see page 107\)](#)

2.5.1.1.3.3 - Erase

2.5.1.1.3.3.1 - CBlob::Erase Method

Erases the byte at uIndex from the blob.

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the byte to erase.

Description

This method erases from the blob the byte at uIndex.

See Also

[Insert](#) (see page 104)

2.5.1.1.3.3.2 - CBlob::Erase Method

Erases uCount bytes at uIndex from the blob.

C++

```
void Erase(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the first byte to erase.
IN unsigned int uCount	The number of bytes to erase.

Description

This method erases from the blob uCount bytes at uIndex.

See Also

[Insert](#) (see page 104)

2.5.1.1.3.4 - CBlob::EraseAll Method

Erases all the bytes from the blob.

C++

```
void EraseAll();
```

Description

This method erases all the bytes from the blob.

See Also

[Insert](#) (see page 104)

2.5.1.1.3.5 - GetAt

2.5.1.1.3.5.1 - CBlob::GetAt Method

Returns a reference to the byte at uIndex.

C++

```
uint8_t& GetAt(IN unsigned int uIndex);
const uint8_t& GetAt(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the byte to retrieve.

Returns

The byte at ulIndex.

Description

This method returns a reference to the byte at ulIndex. It returns a reference to an invalid byte if ulIndex is greater than or equal to the size.

See Also

`operator[]`

2.5.1.1.3.6 - CBlob::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

This method returns the capacity of the blob. The capacity is the number of bytes allocated that are available for use.

2.5.1.1.3.7 - CBlob::GetEndIndex Method

Returns the index of the first unused byte.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused byte.

Description

This method returns the index of the first unused byte. This is one byte passed the last used byte in the blob.

2.5.1.1.3.8 - GetEndIndexPtr**2.5.1.1.3.8.1 - CBlob::GetEndIndexPtr Method**

Returns a pointer to the byte at the end index.

C++

```
uint8_t* GetEndIndexPtr();
const uint8_t* GetEndIndexPtr() const;
```

Returns

NULL if the size equals the capacity, a pointer to the byte at end index otherwise.

Description

This method returns a pointer to the byte at end index in the blob's internal buffer. The memory location returned is one past the last valid byte in the buffer. This method is used when the client wants to modify, insert, or remove data manually.

Warning

- Never delete a pointer acquired through GetEndIndexPtr. The blob is always the owner of the buffer.
- If bytes of data are inserted or removed using this approach, care must be taken to avoid buffer overflow.
- Avoid keeping the pointer returned by this method. Any operation on the blob that can lead to memory allocation makes such a

pointer obsolete.

See Also

[GetFirstIndexPtr \(see page 102\)](#), [Resize \(see page 109\)](#), [GetSize \(see page 103\)](#), [GetCapacity \(see page 101\)](#)

2.5.1.1.3.9 - CBlob::GetFirstIndex Method

Returns the index of the first used byte.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first used byte.

Description

This method returns the index of the first used byte.

2.5.1.1.3.10 - GetFirstIndexPtr

2.5.1.1.3.10.1 - CBlob::GetFirstIndexPtr Method

Returns a pointer to the byte at index zero.

C++

```
uint8_t* GetFirstIndexPtr();
const uint8_t* GetFirstIndexPtr() const;
```

Returns

NULL if the capacity is 0, a pointer to the byte at index zero otherwise.

Description

This method returns a pointer to the byte at index zero in the blob's internal buffer. This method is used when the client wants to modify, insert, or remove data manually.

Warning

- Never delete a pointer acquired through GetFirstIndexPtr. The blob is always the owner of the buffer.
- If bytes of data are inserted or removed using this approach, care must be taken to avoid buffer overflow.
- Avoid keeping the pointer returned by this method. Any operation on the blob that can lead to memory allocation makes such a pointer obsolete.

See Also

[GetEndIndexPtr \(see page 101\)](#), [Resize \(see page 109\)](#), [GetSize \(see page 103\)](#), [GetCapacity \(see page 101\)](#)

2.5.1.1.3.11 - CBlob::GetLastIndex Method

Returns the index of the last used byte.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last used byte.

Description

This method returns the index of the last used byte.

2.5.1.1.3.12 - **CBlob::GetLockCapacity** Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

This method returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 106) is called and decreased each time UnlockCapacity (see page 111) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 108) or ReserveCapacity (see page 109) fails.

2.5.1.1.3.13 - **CBlob::GetSize** Method

Gets the size.

C++

```
unsigned int GetSize() const;
```

Returns

The size of the blob.

Description

This method returns the size of the blob. The size is the number of bytes that are already in use in the blob.

2.5.1.1.3.14 - **CBlob::GetUnreadBits** Method

Gets the number of unread bits.

C++

```
unsigned int GetUnreadBits();
```

Returns

The number of unread bits.

Description

This method retrieves the remaining number of bits to read from the blob.

See Also

ReadBits (see page 107)

2.5.1.1.3.15 - **CBlob::GetUnreadSize** Method

Gets the remaining number of bytes to read from the blob.

C++

```
unsigned int GetUnreadSize();
```

Returns

The remaining number of bytes to read.

Description

This method simply returns the remaining number of bytes to read from the blob.

See Also

Read (see page 106), ReadNoCopy (see page 107)

2.5.1.1.3.16 - **Insert**

2.5.1.1.3.16.1 - CBlob::Insert Method

Inserts the bytes contained in another blob.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN const CBlob& rBlob, IN unsigned int uSrcBlobIndex = 0, IN unsigned int uSize = uBLOB_COPY_ALL);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the byte(s).
IN const CBlob& rBlob	A blob that contains the bytes to insert.
IN unsigned int uSrcBlobIndex = 0	Index of the first byte to insert from rBlob (Default is 0).
IN unsigned int uSize = uBLOB_COPY_ALL	Size in bytes to insert from rBlob (Default is entire blob).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

This method inserts the bytes contained in another blob at uIndex. The capacity is increased if not big enough to accommodate the insertion of the new bytes. If the lock capacity count is non-zero, the insertion fails. When uSrcBlobIndex and uSize are left to default, the content of rBlob is completely inserted. When uSrcBlobIndex and uSize are specified, uSize bytes of rBlob starting at uSrcBlobIndex are inserted.

See Also

Append (see page 98), Push, Enqueue, Erase (see page 100), EraseAll (see page 100)

2.5.1.1.3.16.2 - CBlob::Insert Method

Inserts bytes at the specified index.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN const uint8_t* puData, IN unsigned int uDataSize);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the bytes.
IN const uint8_t* puData	A pointer to the bytes to insert.
IN unsigned int uDataSize	The number of bytes to insert.

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

This method inserts uDataSize bytes of data at uIndex. The bytes to be inserted are pointed to by puData.

If the capacity is insufficient, the size of the buffer is augmented according to the current capacity lock behaviour.

See Also

Append (see page 98), GetLockCapacity (see page 103), LockCapacity (see page 106), UnlockCapacity (see page 111)

2.5.1.1.3.16.3 - CBlob::Insert Method

Inserts one or more bytes.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the byte(s).
IN unsigned int uCount	The number of bytes to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

This method inserts uCount uninitialized bytes at uIndex in the blob. The capacity is increased if not big enough to accommodate the insertion of the new bytes. If the lock capacity count is non-zero, the insertion fails.

See Also

Append ([see page 98](#)), Erase ([see page 100](#)), EraseAll ([see page 100](#))

2.5.1.1.3.16.4 - CBlob::Insert Method

Inserts one or more bytes.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN uint8_t uValue);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the byte(s).
IN unsigned int uCount	The number of bytes to insert.
IN uint8_t uValue	The value with which to initialize inserted bytes.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

This method inserts uCount bytes initialized to uValue at uIndex in the blob. The capacity is increased if not big enough to accommodate the insertion of the new bytes. If the lock capacity count is non-zero, the insertion fails.

See Also

Append ([see page 98](#)), Erase ([see page 100](#)), EraseAll ([see page 100](#))

2.5.1.1.3.17 - CBlob::IsEmpty Method

Checks if the blob is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

This method returns true if the size is 0; in other words, if no bytes are currently used in the blob.

2.5.1.1.3.18 - CBlob::IsFull Method

Checks if the blob is full.

C++

```
bool IsFull() const;
```

Returns

True when the blob is full according to its capacity.

Description

This method returns true when the size of the blob equals its capacity, i.e., there is no more room to add new data to it without allocating more memory.

2.5.1.1.3.19 - CBlob::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

This method locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 111) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 108) or ReserveCapacity (see page 109) fails.

2.5.1.1.3.20 - CBlob::Merge Method

Moves all the byte of another blob to a specific index.

C++

```
mxt_result Merge(IN unsigned int uIndex, INOUT CBlob& rBlob);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the blob.
INOUT CBlob& rBlob	The source blob.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

This method moves all the bytes of another blob to a specific index. The capacity is increased if not big enough to accommodate the insertion of the new bytes. If the lock capacity count is non-zero, the insertion fails.

See Also

Insert (see page 104), Split (see page 110)

2.5.1.1.3.21 - CBlob::PadBits Method

Pads with cleared ('0') bits at the end of the blob.

C++

```
void PadBits(IN unsigned int uSizeInBits);
```

Parameters

Parameters	Description
IN unsigned int uSizeInBits	The number of bits to pad.

Description

This method appends uSizeInBits bits of value 0. It is useful for padding.

See Also

AppendBits (see page 99)

2.5.1.1.3.22 - CBlob::Read Method

Reads bytes starting at the current read index.

C++

```
uint8_t* Read(OUT uint8_t* pReadDest, IN unsigned int uSize);
```

Parameters

Parameters	Description
OUT uint8_t* pReadDest	A pointer where to copy the data to read.

IN unsigned int uSize	The number of bytes to read.
-----------------------	------------------------------

Returns

NULL if the supplied pointer is invalid or if uSize > unread size. A pointer to pReadDest otherwise.

Description

This method reads uSize bytes from the blob into pReadDest. The read index is incremented. The pointer pReadDest is used assuming that the copy of uSize bytes does not produce overflow.

See Also

ReadNoCopy (see page 107), GetUnreadSize (see page 103)

2.5.1.1.3.23 - CBlob::ReadBits Method

Reads bits at the current read index.

C++

```
void ReadBits(OUT uint8_t* puReadDest, IN unsigned int uSizeInBits, IN unsigned int uStartingBit = 0);
```

Parameters

Parameters	Description
OUT uint8_t* puReadDest	A pointer where to copy the bits to read.
IN unsigned int uSizeInBits	The number of bits to read.
IN unsigned int uStartingBit = 0	The position where to store the first bit to read in puReadDest. The position should be included in the range 0 - 7 where 0 is the MSB.

Description

This method reads uSizeInBits bits of data from the blob at the current read index. Successive calls to this method result in reading the bits just after the bits previously read. Any call to Read (see page 106) or ReadNoCopy (see page 107) causes a realignment on a one byte boundary.

- It is the caller's responsibility to be sure there are enough unread bits in the blob with the GetUnreadBits (see page 103) method.
- If there are less than uSizeInBits bits to read, only the available ones are copied, and the read index is moved to the end.

See Also

AppendBits (see page 99)

2.5.1.1.3.24 - CBlob::ReadNoCopy Method

Reads bytes starting at the current read index.

C++

```
const uint8_t* ReadNoCopy(IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uSize	The number of bytes to read.

Returns

NULL if uSize > unread size. The pointer to the current read index in the internal buffer otherwise.

Description

This method reads uSize bytes from the blob. The pointer to the byte at the current read index in the internal buffer is returned. The read index is incremented.

Warning

A pointer to the internal buffer is returned and can be invalidated by any call to methods that modify its internal buffer.

See Also

Read (see page 106), GetUnreadSize (see page 103)

2.5.1.1.3.25 - CBlob::RealignAppendBits Method

Pads the needed number of bits at the end of the blob so that it is realigned on a multiple of the given number of bits.

C++

```
void RealignAppendBits(IN unsigned int uBitsMultiple = 8);
```

Parameters

Parameters	Description
IN unsigned int uBitsMultiple = 8	The number of bits on which to realign the append. This parameter defaults to 8 bits (1 byte).

Description

Call this member function to pad the needed number of bits at the end of the blob so that the next append is realigned on a multiple of the specified number of bits.

The needed number of bits is between 0 and uBitsMultiple-1.

See Also

AppendBits (see page 99), PadBits (see page 106)

2.5.1.1.3.26 - CBlob::RealignReadBits Method

Increments (if needed) the read index so that it is a multiple of the given number of bits.

C++

```
void RealignReadBits(IN unsigned int uBitsMultiple = 8);
```

Parameters

Parameters	Description
IN unsigned int uBitsMultiple = 8	The number of bits on which to realign the read index. This parameter defaults to 8 bits (1 byte).

Description

Call this member function to skip the needed number of bits, starting from the current read index so that it is realigned on a multiple of the specified number of bits.

The needed number of bits is between 0 and uBitsMultiple-1.

See Also

ReadBits (see page 107), SkipBits (see page 110)

2.5.1.1.3.27 - CBlob::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

This method reduces the capacity of the blob. The capacity is reduced to uDownToCapacity only if uDownToCapacity is below the current capacity.

The method fails if the lock capacity count is not 0.

2.5.1.1.3.28 - CBlob::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

This method increases the capacity of the blob. The capacity is increased to uUpToCapacity only if uUpToCapacity is greater than the current capacity. The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.1.1.3.29 - Resize

2.5.1.1.3.29.1 - CBlob::Resize Method

Resizes the blob.

C++

```
mxt_result Resize(IN uint8_t* pEnd);
```

Parameters

Parameters	Description
IN uint8_t* pEnd	The pointer to the wanted end byte in the blob.

Returns

resS_OK resFE_INVALID_ARGUMENT

Description

This method resizes the number of bytes in use in the blob according to pEnd. The end byte in the blob is not valid, it is one byte passed the last valid byte. The supplied pEnd must be within the boundaries of the blob's buffer. This method is useful when the blob's internal buffer is used directly.

See Also

[GetFirstIndexPtr](#) (see page 102), [GetEndIndexPtr](#) (see page 101)

2.5.1.1.3.29.2 - CBlob::Resize Method

Resizes the blob.

C++

```
mxt_result Resize(IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uSize	The size representing used bytes in the blob.

Returns

resS_OK resFE_INVALID_ARGUMENT

Description

This method resizes to uSize the number of bytes in use in the blob. The parameter uSize must fit into the blob's capacity. This method is useful when the blob's internal buffer is used directly.

See Also

[GetFirstIndexPtr](#) (see page 102), [GetEndIndexPtr](#) (see page 101)

2.5.1.1.3.30 - CBlob::SetReadIndex Method

Sets the current read index.

C++

```
mxt_result SetReadIndex(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The read index in the blob's buffer.

Returns

[resS_OK](#) [resFE_INVALID_ARGUMENT](#)

Description

This method sets the read index to uIndex in the blob's buffer. Index is zero-based in the blob buffer.

See Also

[Read](#) (see page 106), [ReadNoCopy](#) (see page 107), [GetUnreadSize](#) (see page 103)

2.5.1.1.3.31 - CBlob::SkipBits Method

Skips bits before reading.

C++

```
void SkipBits(IN unsigned int uSizeInBits);
```

Parameters

Parameters	Description
IN unsigned int uSizeInBits	The number of bits to skip.

Description

This method skips uSizeInBits bits starting from the current read index. In other words, the read index is incremented uSizeInBits bits.

See Also

[ReadBits](#) (see page 107)

2.5.1.1.3.32 - CBlob::Split Method

Moves some bytes from this blob to another starting from a specific index.

C++

```
mxt_result Split(IN unsigned int uIndex, OUT CBlob& rBlob);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the bytes.
OUT CBlob& rBlob	The destination blob.

Returns

[resS_OK](#) [resFE_OUT_OF_MEMORY](#)

Description

This method moves some bytes from this blob to another starting from a specific index. The destination blob is emptied first. The capacity is increased if not big enough to accommodate the insertion of the new bytes. If the lock capacity count is non-zero, the insertion fails.

See Also

Insert (see page 104), Merge (see page 106)

2.5.1.1.3.33 - CBlob::Swap Method

This method exchanges the position of two bytes.

C++

```
void Swap(IN unsigned int uFirstIndex, IN unsigned int uSecondIndex);
```

Parameters

Parameters	Description
IN unsigned int uFirstIndex	The index of the first byte to swap.
IN unsigned int uSecondIndex	The index of the second byte to swap.

Description

This method exchanges the position of two bytes.

2.5.1.1.3.34 - CBlob::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

This method unlocks the capacity of the blob. The lock capacity count is a counter that is increased each time LockCapacity (see page 106) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 108) or ReserveCapacity (see page 109) fails.

2.5.1.1.4 - Operators**2.5.1.1.4.1 - CBlob::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the two blobs are different, false otherwise

Description

Verifies whether or not the left hand blob is equal to the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.1.4.2 - []**2.5.1.1.4.2.1 - CBlob::[] Operator**

Returns a reference to the byte at ulIndex.

C++

```
const uint8_t& operator [](IN unsigned int ulIndex) const;
uint8_t& operator [](IN unsigned int ulIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of a byte.

Returns

The byte at uIndex.

Description

This method returns a reference to the byte at uIndex. It returns a reference to an invalid byte if uIndex is greater than or equal to the size.

See Also

GetAt (see page 100)

2.5.1.1.4.3 - CBlob::< Operator

Less than operator.

C++

```
bool operator <(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the left hand blob is less than the right hand blob, false otherwise

Description

Verifies that the left hand blob is less than the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.1.4.4 - CBlob::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the left hand blob is less than or equal to the right hand blob, false otherwise

Description

Verifies that the left hand blob is less than or equal to the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.1.4.5 - CBlob::= Operator

Assignment operator.

C++

```
CBlob& operator =(IN const CBlob& rBlob);
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

A reference to this CBlob (see page 95) instance.

Description

Assignment operator.

2.5.1.1.4.6 - CBlob::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the two blobs are equal, false otherwise

Description

Verifies whether or not the left hand blob is equal to the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.1.4.7 - CBlob::> Operator

Greater than operator.

C++

```
bool operator >(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the left hand blob is greater than the right hand blob, false otherwise

Description

Verifies that the left hand blob is greater than the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.1.4.8 - CBlob::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN const CBlob& rBlob	A reference to the source blob.

Returns

True if the left hand blob is greater than or equal to the right hand blob, false otherwise

Description

Verifies that the left hand blob is greater than or equal to the right hand blob. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.1.2 - CBlockAllocator Class

Manages a list of free blocks.

Class Hierarchy

```
CBlockAllocator
```

C++

```
class CBlockAllocator;
```

Description

CBlockAllocator manages a list of free blocks. It supports four different modification methods: Pop (see page 116), Push (see page 116), Reduce (see page 116), and Reserve (see page 116).

The Pop (see page 116) method retrieves a free block from the list. A new block is allocated if the list of free blocks is empty.

The Push (see page 116) method returns a block to the list of free blocks. The memory associated with the block is not released.

The Reserve (see page 116) method exists for performance reasons. When Reserve (see page 116) is not called, each call to Pop (see page 116) forces a memory allocation if the list of free blocks is empty. Reserve (see page 116) should be called to make sure a minimum number of free blocks are contained within the list. If the list does not contain enough blocks, they are allocated and added to the list.

The Reduce (see page 116) method exists for conditions where the list of free blocks is too large. Reduce (see page 116) can be called to reduce the number of free blocks in the list to a maximum level.

For example:

Calling Pop (see page 116) when the list contains 0 free block generates an allocation of 1 block.

Calling Pop (see page 116) when the list contains more than 0 free blocks does not generate an allocation.

Calling Push (see page 116) adds the block to the list of free blocks.

Calling Reserve (see page 116)(n) when the list already contains n or more free blocks does nothing.

Calling Reserve (see page 116)(n) when the list contains less than n free blocks forces the allocation of the missing blocks.

Calling Reduce (see page 116)(n) when the list already contains n or less free blocks does nothing.

Calling Reduce (see page 116)(n) when the list contains more than n free blocks forces the releasing of the extra blocks.

Location

Cap/CBlockAllocator.h

Constructors

Constructor	Description
CBlockAllocator (see page 115)	Copy constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CBlockAllocator (see page 115)	Default destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 117)	Different than operator.
 == (see page 117)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
• GetBlockCount (See page 116)	Retrieves the number of free blocks available.
• Pop (See page 116)	Retrieves a block from the list of free blocks. Allocates it if the list is empty.
• Push (See page 116)	Returns a block to the list of free blocks.
• Reduce (See page 116)	Reduces the list of free blocks to a maximum of uDownToBlockCount blocks.
• Reserve (See page 116)	Increments the list of free blocks to a minimum of uUpToBlockCount blocks.

Legend

•	Method
▼	virtual

2.5.1.2.1 - Constructors

2.5.1.2.1.1 - CBlockAllocator

2.5.1.2.1.1.1 - CBlockAllocator::CBlockAllocator Constructor

Copy constructor.

C++

```
CBlockAllocator(IN const CBlockAllocator& rBlockAllocator);
```

Parameters

Parameters	Description
IN const CBlockAllocator& rBlockAllocator	Reference to a CblockAllocator to use as the allocator.

Description

Copy constructor.

2.5.1.2.1.1.2 - CBlockAllocator::CBlockAllocator Constructor

Constructor

C++

```
CBlockAllocator(IN unsigned int uBlockSize, IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN unsigned int uBlockSize	Size of the blocks that are allocated.
IN IAllocator* pAllocator = NULL	Pointer to an IAllocator (See page 156).

Description

Constructor. Sets the size of the blocks to allocate and may set the allocator to use.

2.5.1.2.2 - Destructors

2.5.1.2.2.1 - CBlockAllocator::~CBlockAllocator Destructor

Default destructor.

C++

```
virtual ~CBlockAllocator();
```

Description

Default destructor.

2.5.1.2.3 - Methods

2.5.1.2.3.1 - CBlockAllocator::GetBlockCount Method

Retrieves the number of free blocks available.

C++

```
unsigned int GetBlockCount() const;
```

Returns

The number of free blocks available.

Description

Retrieves the number of free blocks available.

2.5.1.2.3.2 - CBlockAllocator::Pop Method

Retrieves a block from the list of free blocks. Allocates it if the list is empty.

C++

```
void* GO_Pop();
```

Returns

A pointer to the block. NULL if it fails.

Description

Retrieves a block from the list of free blocks. Allocates it if the list is empty. Returns NULL if out of memory. Calling Reserve (see page 116) before Pop is a good way to ensure that blocks are available.

2.5.1.2.3.3 - CBlockAllocator::Push Method

Returns a block to the list of free blocks.

C++

```
virtual void Push(IN TOA void* pBlock);
```

Parameters

Parameters	Description
IN TOA void* pBlock	A pointer to the block that must be returned to the list of free blocks.

Description

Returns a block to the list of free blocks.

2.5.1.2.3.4 - CBlockAllocator::Reduce Method

Reduces the list of free blocks to a maximum of uDownToBlockCount blocks.

C++

```
void Reduce(IN unsigned int uDownToBlockCount);
```

Parameters

Parameters	Description
IN unsigned int uDownToBlockCount	The maximum number of free blocks allowed within the list.

Description

Reduces the list of free blocks to a maximum of uDownToBlockCount blocks.

2.5.1.2.3.5 - CBlockAllocator::Reserve Method

Increments the list of free blocks to a minimum of uUpToBlockCount blocks.

C++

```
bool Reserve(IN unsigned int uUpToBlockCount);
```

Parameters

Parameters	Description
IN unsigned int uUpToBlockCount	The minimum number of free blocks allowed within the list.

Returns

True if Reserve is successful. False otherwise.

Description

Increments the list of free blocks to a minimum of uUpToBlockCount blocks.

2.5.1.2.4 - Operators**2.5.1.2.4.1 - CBlockAllocator::!= Operator**

Different than operator.

C++

```
bool operator !=(const CBlockAllocator& rAllocator) const;
```

Parameters

Parameters	Description
const CBlockAllocator& rAllocator	Reference to the block allocator to compare.

Returns

True if both CBlockAllocators are different, false otherwise.

Description

Different than operator. Checks if both block allocators are different.

2.5.1.2.4.2 - CBlockAllocator::== Operator

Comparison operator.

C++

```
bool operator ==(const CBlockAllocator& rAllocator) const;
```

Parameters

Parameters	Description
const CBlockAllocator& rAllocator	Reference to the block allocator to compare.

Returns

True if both CBlockAllocators are equal, false otherwise.

Description

Comparison operator. Checks if both block allocators are equal.

2.5.1.3 - CMarshaler Class

Class used to store objects.

Class Hierarchy**C++**

```
class CMarshaler;
```

Description

The CMarshaller class is used to store objects. It is designed to rapidly insert and extract objects stored within. All elements are first in first out. It is up to the application to be prepared to receive any kind of data when getting an object.

Objects stored inside the marshaler are put into a preallocated pool.

Constructors

Constructor	Description
CMarshaler (see page 119)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CMarshaler (see page 119)	Destructor.

Legend

	Method
--	--------

Operators

Operator	Description
<< (see page 120)	Inserts an object into the marshaler.
= (see page 121)	Assignment operator
>> (see page 121)	Extracts an object from the marshaler.

Legend

	Method
--	--------

Methods

Method	Description
Clear (see page 119)	Clears the marshaler.
IsEmpty (see page 119)	Checks whether or not the marshaler is empty.
Load (see page 120)	Loads data from the marshaler.
Store (see page 120)	Stores variable length data into the marshaler.

Legend

	Method
--	--------

Functions

Function	Description
<< (see page 120)	Inserts an object into the marshaler.
<< (see page 118)	&>
>> (see page 121)	Extracts an object from the marshaler.
>> (see page 118)	&>

2.5.1.3.1 - << Function

&>

C++

```
template <class _Type> CMarshaler& operator <<(IN CMarshaler& rMarshaler, IN const CSharedPtr<_Type>& rstSharedPtr);
```

2.5.1.3.2 - >> Function

&>

&>

C++

```
template <class _Type> CMarshaler& operator >>(IN CMarshaler& rMarshaler, OUT CSharedPtr<_Type>& rstSharedPtr);
```

2.5.1.3.3 - Constructors

2.5.1.3.3.1 - CMarshaler

2.5.1.3.3.1.1 - CMarshaler::CMarshaler Constructor

Constructor.

C++

```
CMarshaler();
```

Description

Constructor.

2.5.1.3.3.1.2 - CMarshaler::CMarshaler Constructor

Copy constructor.

C++

```
CMarshaler(const CMarshaler& rMarshaler);
```

Parameters

Parameters	Description
const CMarshaler& rMarshaler	Reference to the CMarshaler object to copy.

Description

Copy constructor.

2.5.1.3.4 - Destructors

2.5.1.3.4.1 - CMarshaler::~CMarshaler Destructor

Destructor.

C++

```
~CMarshaler();
```

Description

Destructor.

2.5.1.3.5 - Methods

2.5.1.3.5.1 - CMarshaler::Clear Method

Clears the marshaler.

C++

```
void Clear();
```

Description

Clears all items in the marshaler.

2.5.1.3.5.2 - CMarshaler::IsEmpty Method

Checks whether or not the marshaler is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the marshaler is empty, false otherwise.

Description

Checks whether or not the CMarshaler (see page 117) object is empty.

2.5.1.3.5.3 - CMarshaler::Load Method

Loads data from the marshaler.

C++

```
bool Load(OUT void* pData, IN unsigned int uCapacity);
```

Parameters

Parameters	Description
OUT void* pData	Pointer to the data that will be loaded.
IN unsigned int uCapacity	Maximum size for the data that will be loaded.

Description

Loads data from the marshaler.

2.5.1.3.5.4 - CMarshaler::Store Method

Stores variable length data into the marshaler.

C++

```
void Store(IN const void* pData, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN const void* pData	The data to be stored in the marshaler.
IN unsigned int uSize	The size of the data.

Description

Stores variable length data into the marshaler.

2.5.1.3.6 - Operators

2.5.1.3.6.1 - <<

2.5.1.3.6.1.1 - CMarshaler::<< Operator

Inserts an object into the marshaler.

C++

```
template <class _Type> CMarshaler& operator <<(IN CMarshaler& rMarshaler, IN const _Type* pPtr);
CMarshaler& operator <<(IN bool data);
CMarshaler& operator <<(IN char data);
CMarshaler& operator <<(IN unsigned char data);
CMarshaler& operator <<(IN signed char data);
CMarshaler& operator <<(IN unsigned short data);
CMarshaler& operator <<(IN signed short data);
CMarshaler& operator <<(IN unsigned int data);
CMarshaler& operator <<(IN signed int data);
CMarshaler& operator <<(IN unsigned long data);
CMarshaler& operator <<(IN signed long data);
CMarshaler& operator <<(IN unsigned __int64 data);
CMarshaler& operator <<(IN __int64 data);
CMarshaler& operator <<(IN unsigned long long data);
CMarshaler& operator <<(IN signed long long data);
CMarshaler& operator <<(IN const float& rData);
CMarshaler& operator <<(IN const double& rData);
CMarshaler& operator <<(IN const long double& rData);
```

Parameters

Parameters	Description
IN bool data	Object to insert into the marshaler.

Description

Inserts an object into the marshaler. It is possible to marshal classes into the CMarshaler (see page 117) object. To do this, the <<

operators must be added to the class to marshal for a CMarshaler (see page 117). Please refer to the CBlob (see page 95) implementation in CBlob.h and CBlob.cpp for an example of how to implement the << operator.

A templated version of this operator can marshal a pointer to any type.

See Also

CMarshaler::operator>> (see page 121)

2.5.1.3.6.2 - CMarshaler::= Operator

Assignment operator

C++

```
CMarshaler& operator =(const CMarshaler& rFrom);
```

2.5.1.3.6.3 - >>

2.5.1.3.6.3.1 - CMarshaler::>> Operator

Extracts an object from the marshaler.

C++

```
template <class _Type> CMarshaler& operator >>(IN CMarshaler& rMarshaler, OUT _Type*& pPtr);
CMarshaler& operator >>(OUT bool& rData);
CMarshaler& operator >>(OUT char& rData);
CMarshaler& operator >>(OUT unsigned char& rData);
CMarshaler& operator >>(OUT signed char& rData);
CMarshaler& operator >>(OUT unsigned short& rData);
CMarshaler& operator >>(OUT signed short& rData);
CMarshaler& operator >>(OUT unsigned int& rData);
CMarshaler& operator >>(OUT signed int& rData);
CMarshaler& operator >>(OUT unsigned long& rData);
CMarshaler& operator >>(OUT signed long& rData);
CMarshaler& operator >>(OUT unsigned __int64& rData);
CMarshaler& operator >>(OUT __int64& rData);
CMarshaler& operator >>(OUT unsigned long long& rData);
CMarshaler& operator >>(OUT signed long long& rData);
CMarshaler& operator >>(OUT float& rData);
CMarshaler& operator >>(OUT double& rData);
CMarshaler& operator >>(OUT long double& rData);
```

Parameters

Parameters	Description
OUT bool& rData	Object extracted from the marshaler.

Description

Extracts an object from the marshaler. It is possible to marshal classes into the CMarshaler (see page 117) object. A marshalled class must be extracted at some point this is the operator used to do this.

A templated version of this operator can marshal a pointer to any type.

See Also

CMarshaler::operator<< (see page 120)

2.5.1.4 - CMemoryQueue Class

Implements a fixed size queue containing memory blocks of variable length.

Class Hierarchy

```
CMemoryQueue
```

C++

```
class CMemoryQueue;
```

Description

This class implements a fixed size queue containing memory blocks of variable length. It is possible to set the alignment for the start of the allocated block in memory and is usually required for structures.

Warning

The queue is ALWAYS the owner of the allocated memory block. Never keep or access a pointer to the memory block after PushFinalize (see page 126) or PopFinalize (see page 125) is called. After these methods are called, the pointer to the memory block becomes invalid and all references to it must be set to NULL. If the data is to be used and accessed later, you must make a copy of it in another memory area.

Published Interface:

Initialize (see page 123): Allocates space for the queue and initializes its internal variables.

Uninitialize (see page 126): Deallocates space for the queue.

Clear (see page 123): Resets the read and write pointers to the beginning of the queue.

GetBlockSize (see page 123): Returns the size of a block.

IsBlockInQueue (see page 124): Returns if a block is located within the queue.

IsEmpty (see page 124): Returns true if the queue is empty.

Push (see page 125): Makes sure space is available at the end of the queue and returns a pointer to the allocated memory block.

PushFinalize (see page 126): Advances the write pointer to the next free memory zone. WARNING: The user MUST always call PushFinalize (see page 126) after memory block access is completed in order to complete the allocation.

PushAbort (see page 125): Ignores the last Push (see page 125) call. Always call this method if the push should be undone.

Pop (see page 124): Returns the oldest block within the queue and its size.

PopFinalize (see page 125): Frees the memory zone and advances the read pointer to the next memory block. WARNING: The user MUST always call PopFinalize (see page 125) after memory block access is completed in order to release and free the block.

PopAbort (see page 125): Ignores the last Pop (see page 124) call. Always call this method if the pop should be undone.

Location

Architecture/CMemoryQueue.h

Constructors

Constructor	Description
◆ CMemoryQueue (see page 123)	Constructor.

Legend

◆	Method
---	--------

Destructors

Destructor	Description
◆ ~CMemoryQueue (see page 123)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
◆ Clear (see page 123)	Resets the read and write pointers to the beginning of the queue.
◆ GetBlockSize (see page 123)	Returns the size of a block.
◆ Initialize (see page 123)	Allocates space for the queue and initializes its internal variables.
◆ IsBlockInQueue (see page 124)	Returns if a block is located within the queue.
◆ IsEmpty (see page 124)	Returns true if the queue is empty.
◆ Pop (see page 124)	Returns a pointer to the first block of the queue.
◆ PopAbort (see page 125)	Aborts the previous call to Pop (see page 124).
◆ PopFinalize (see page 125)	Finalizes the previous call to Pop (see page 124).
◆ Push (see page 125)	Makes sure space is available at the end of the queue and returns a pointer to the allocated memory block.
◆ PushAbort (see page 125)	Aborts the previous call to Push (see page 125).
◆ PushFinalize (see page 126)	Finalizes the previous call to Push (see page 125).
◆ Uninitialize (see page 126)	Deallocates space for the queue.

Legend**2.5.1.4.1 - Constructors****2.5.1.4.1.1 - CMemoryQueue::CMemoryQueue Constructor**

Constructor.

C++

```
CMemoryQueue();
```

Description

Constructor.

2.5.1.4.2 - Destructors**2.5.1.4.2.1 - CMemoryQueue::~CMemoryQueue Destructor**

Destructor.

C++

```
virtual ~CMemoryQueue();
```

Description

Destructor.

2.5.1.4.3 - Methods**2.5.1.4.3.1 - CMemoryQueue::Clear Method**

Resets the read and write pointers to the beginning of the queue.

C++

```
void Clear();
```

Description

Resets the read and write pointers to the beginning of the queue.

2.5.1.4.3.2 - CMemoryQueue::GetBlockSize Method

Returns the size of a block.

C++

```
unsigned int GetBlockSize(IN uint8_t* puBlock);
```

Parameters

Parameters	Description
IN uint8_t* puBlock	Pointer to a block of memory.

Returns

The size of the memory block.

Description

Returns the size of a block.

2.5.1.4.3.3 - CMemoryQueue::Initialize Method

Allocates space for the queue and initializes its internal variables.

C++

```
bool Initialize(IN unsigned int uQueueSize, IN unsigned int uMemoryAlignment = sizeof(unsigned int));
```

Parameters

Parameters	Description
IN unsigned int uQueueSize	The maximum queue size.
IN unsigned int uMemoryAlignment = sizeof(unsigned int)	The value (multiple) on which write and read pointers are aligned. Default is set to sizeof(unsigned int).

Returns

Returns true if the dynamic allocation of space for the queue succeeds.

Description

Allocates space for the queue and initializes its internal variables.

2.5.1.4.3.4 - CMemoryQueue::IsBlockInQueue Method

Returns if a block is located within the queue.

C++

```
bool IsBlockInQueue(IN uint8_t* puBlock);
```

Parameters

Parameters	Description
IN uint8_t* puBlock	Pointer to a block of memory.

Returns

True if the block is located within the queue, false otherwise.

Description

Returns if a block is located within the queue.

2.5.1.4.3.5 - CMemoryQueue::IsEmpty Method

Returns true if the queue is empty.

C++

```
bool IsEmpty();
```

Returns

True when empty, false otherwise.

Description

Returns true if the queue is empty.

2.5.1.4.3.6 - CMemoryQueue::Pop Method

Returns a pointer to the first block of the queue.

C++

```
uint8_t* Pop(OUT unsigned int* puSize);
```

Parameters

Parameters	Description
OUT unsigned int* puSize	Contains the size of the retrieved block. If the size is unnecessary, NULL can be passed.

Returns

Returns the allocated block, NULL if the queue is empty.

Description

This method returns the oldest block within the queue and its size. The user MUST always call PopFinalize (see page 125) when it is finished with the block. This releases the memory associated with the block.

2.5.1.4.3.7 - CMemoryQueue::PopAbort Method

Aborts the previous call to Pop (see page 124).

C++

```
void PopAbort();
```

Description

This method must be called if the user wants to abort the pop procedure.

2.5.1.4.3.8 - CMemoryQueue::PopFinalize Method

Finalizes the previous call to Pop (see page 124).

C++

```
void PopFinalize();
```

Description

Advances the write pointer to the next free memory zone.

Warning

Never keep or access a pointer to the memory block after PushFinalize (see page 126) or PopFinalize is called. After these methods are called, the queue becomes the owner of the memory block and any external pointers must be set to NULL. If the data is to be used and accessed later, you must make a copy of it in another memory area.

Notes

The user MUST always call PushFinalize (see page 126) after the memory block access is completed in order to complete the allocation.

2.5.1.4.3.9 - CMemoryQueue::Push Method

Makes sure space is available at the end of the queue and returns a pointer to the allocated memory block.

C++

```
uint8_t* Push(IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uSize	The maximum size of the block to be pushed. This size may be reduced when calling PushFinalize (see page 126).

Returns

Returns the allocated block, NULL if the queue is full.

Description

This method makes sure space is available at the end of the queue to allocate the block. The user MUST always call PushFinalize (see page 126) when he is finished with the block. If the user wants to undo the push, he must call PushAbort (see page 125). This method returns a pointer to the allocated block.

Notes

The situation in which the Write pointer would reach the read pointer in order to be able to distinguish a full queue from an empty one must be avoided.

2.5.1.4.3.10 - CMemoryQueue::PushAbort Method

Aborts the previous call to Push (see page 125).

C++

```
void PushAbort();
```

Description

The method must be called if the user wants to abort the push procedure.

2.5.1.4.3.11 - CMemoryQueue::PushFinalize Method

Finalizes the previous call to Push (see page 125).

C++

```
void PushFinalize(IN unsigned int uSize = 0);
```

Parameters

Parameters	Description
IN unsigned int uSize = 0	Offers the possibility to reduce the size of the pushed block. If 0, it keeps the size received in Push (see page 125).

Description

Advances the write pointer to the next free memory zone.

Warning

uSize cannot be greater than the uSize passed in the previous call to Push (see page 125).

Never keep or access a pointer to the memory block after PushFinalize or PopFinalize (see page 125) is called. After these methods are called, the queue becomes the owner of the memory block and any external pointer must be set to NULL. If the data is to be used and accessed later, you must make a copy of it in another memory area.

Notes

The user MUST always call PushFinalize after memory block access is completed in order to complete the allocation.

2.5.1.4.3.12 - CMemoryQueue::Uninitialize Method

Deallocates space for the queue.

C++

```
void Uninitialize();
```

Description

Deallocates space for the queue.

2.5.1.5 - CString Class

Class used to encapsulate a c-style string.

Class Hierarchy

```
CString
```

C++

```
class CString;
```

Description

The CString class is used to encapsulate a c-style string. It has a resizable buffer to hold the string. The buffer's capacity can be modified automatically (insert, append, etc.) or explicitly (ReduceCapacity (see page 136) and ReserveCapacity (see page 137)).

Internal buffer struct diagram

+	-----	-----	-----	-----	-----	-----	-----
	RefCount	Capacity	Size	Array of chars	\0	
+	-----	-----	-----	-----	-----	-----	-----
2/4 Bytes	2 Bytes	2 Bytes	n Bytes				

Notes

- The RefCount size depends on whether MXD_ATOMIC_NATIVE_ENABLE_SUPPORT (see page 276) is enabled and the support

for 16 bit native atomic operations on the current platform/architecture. See CAtomicOperations (see page 515) for more information.

- The minimal size for a new CString object is 7/9 bytes + the size of the internal pointer.
- The maximal size for a CString object is uCSTRING_MAX_VALID_SIZE bytes + the size of the internal pointer.
- The size for a CString copy (RefCount scheme enabled) is the size of the internal pointer.

Memory management: The memory used to store the characters in a CString is allocated with MX_NEW_ARRAY (see page 510). By default, MX_NEW_ARRAY (see page 510) redirects the allocation to standard new operator. If a more efficient memory allocation algorithm is required, refer to the documentation related to MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

The default capacity of a CString is 0 bytes. The CString is autogrow, but to improve performance it is recommended to call ReserveCapacity (see page 137) with the right capacity before use.

The following two sections describe the reference counting mechanism available within the CString class.

Synchronized Reference Counting:

When MXD_STRING_DISABLE_REFCOUNT (see page 307) is NOT defined, the reference count scheme is enabled. Reference Counting is used when multiple CString objects can refer to the same string. For example, the two code lines below take advantage of the RefCount scheme to save memory space.

```
CString strCopy1 = strInitial; CString strCopy2(strInitial);
```

In this example, no new memory is allocated for strCopy1 and strCopy2, except for the four bytes required for the pointer data member. The final RefCount value is three, so any of the three CString objects are aware that they are sharing an internal buffer with other CString instances. The access to the reference counting is synchronized with a single static mutex. When the Ref count is locked for a CString, no other CString can modify its internal buffer. This way, there is no problem when copying a string from one thread to another. **THIS DOES NOT MEAN THAT IT IS SAFE TO (see page 62) SHARE POINTERS TO (see page 62) STRINGS BETWEEN THREADS!** It only means that the reference counting scheme will not cause any problems when a string is shared between two or more threads by using the "=" operator or the copy constructor.

The RefCount value is equal to "0" for an uninitialized CString object.

No Reference Counting:

When MXD_STRING_DISABLE_REFCOUNT (see page 307) is defined, the reference counting scheme is disabled. Any call to the copy constructor or the "=" operator (the two places where the RefCount can be increased) allocates a memory storage for the new CString object. Thus, the RefCount data member always has a value of "0".

See Also

[MXD_STRING_DISABLE_REFCOUNT \(see page 307\)](#), [MXD_uCSTRING_BLOCK_LENGTH \(see page 315\)](#), [CAtomicOperations \(see page 515\)](#)

Constructors

Constructor	Description
 CString (see page 130)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CString (see page 131)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 139)	Checks whether or not two CString objects are different.
 [] (see page 139)	Direct Access operator (read only).
 + (see page 140)	Appends the CString's content to the end of the current CString.
 += (see page 141)	Appends a character to the end of the current CString.

-♦ < (see page 142)	Checks whether or not the left-hand CString is lower than the right-hand CString.
-♦ <= (see page 142)	Checks whether or not the left-hand CString is less than or equal to the right-hand CString.
-♦ = (see page 143)	Assigns a CString to the current CString object.
-♦ == (see page 144)	Checks whether or not two CString objects are identical.
-♦ > (see page 145)	Checks whether or not the left-hand CString is greater than the right-hand CString.
-♦ >= (see page 145)	Checks whether or not the left-hand CString is greater than or equal to the right-hand CString.

Legend

	Method
--	--------

Methods

Method	Description
-♦ Append (see page 131)	Appends a c-style sub string to the end of the current CString.
-♦ CaseInsCmp (see page 131)	Case insensitive comparison between two CString.
-♦ CStr (see page 132)	Returns a pointer to the internal C-style string.
-♦ Erase (see page 132)	Erases one character from the current CString.
-♦ EraseAll (see page 133)	Erases all the characters from the current CString object.
-♦ FindSubstring (see page 133)	Finds the occurrence of a CString in the current CString.
-♦ Format (see page 134)	Formats arguments following pszFormat specifications.
-♦ FormatV (see page 134)	Formats a variable argument list following pszFormat specifications.
-♦ GetAt (see page 134)	Gets one character from the current CString.
-♦ GetBuffer (see page 134)	Returns a pointer to CString's internal buffer.
-♦ GetCapacity (see page 134)	Returns the capacity of the current CString.
-♦ GetSize (see page 135)	Returns the size of the current CString.
-♦ Insert (see page 135)	Inserts a character into the current CString.
-♦ IsEmpty (see page 136)	Checks whether or not CString's internal buffer is empty.
-♦ ReduceCapacity (see page 136)	Reduces the capacity of CString's internal buffer.
-♦ ReserveCapacity (see page 137)	Augments the capacity of CString's internal buffer.
-♦ Resize (see page 137)	Resizes the CString.
-♦ SetAt (see page 137)	Sets one character from the current CString.
-♦ ToLowerCase (see page 138)	Converts to lowercase the contents of the current CString.
-♦ ToUpperCase (see page 138)	Converts to uppercase the contents of the current CString.
-♦ TrimBothSide (see page 138)	Removes all the leading and trailing characters of the specified type.
-♦ TrimLeftSide (see page 138)	Removes all the leading characters of the specified type.
-♦ TrimRightSide (see page 138)	Removes all the trailing characters of the specified type.

Legend

	Method
--	--------

Functions

Function	Description
+ (see page 128)	Appends a CString to the end of a c-style string.
+ (see page 129)	Appends a CString to the end of a character.
<< (see page 129)	Marshaling input operator.
>> (see page 129)	Marshaling output operator.

2.5.1.5.1 - + Function

Appends a CString (see page 126) to the end of a c-style string.

C++

```
CString operator +(IN const char* pszLhs, IN const CString& strRhs);
```

Parameters

Parameters	Description
IN const char* pszLhs	Left-hand c-style string.
IN const CString& strRhs	Right-hand CString (see page 126) operand.

Returns

A CString (see page 126) that is the result of the sum of both operands.

Description

See CString::operator+(IN const CString&) const for more details.

See Also

CString::operator+(IN const CString&) const (see page 140)

2.5.1.5.2 - + Function

Appends a CString (see page 126) to the end of a character.

C++

```
CString operator +(IN const char c, IN const CString& strRhs);
```

Parameters

Parameters	Description
IN const char c	Left-hand character.
IN const CString& strRhs	Right-hand CString (see page 126) operand.

Returns

A CString (see page 126) that is the result of the sum of both operands.

Description

See CString::operator+(IN const CString&) const for more details.

See Also

CString::operator+(IN const CString&) const (see page 140)

2.5.1.5.3 - << Function

Marshaling input operator.

C++

```
CMarshaler& operator <<(IN CMarshaler& rMarshaler, IN const CString& rStr);
```

Parameters

Parameters	Description
IN CMarshaler& rMarshaler	CMarshaler (see page 117) object into which CString (see page 126) is inserted.
IN const CString& rStr	Left-hand CString (see page 126) to be inserted.

Returns

Reference to the CMarshaler object.

Description

Inserts the CString (see page 126) "rStr" into the CMarshaler (see page 117) object "rMarshaler".

2.5.1.5.4 - >> Function

Marshaling output operator.

C++

```
CMarshaler& operator >>(IN CMarshaler& rMarshaler, IN CString& rStr);
```

Parameters

Parameters	Description
IN CMarshaler& rMarshaler	CMarshaler (see page 117) object containing a c-style string.
IN CString& rStr	CString (see page 126) object into which the c-style string from CMarshaler (see page 117) is inserted.

Returns

Reference to the CMarshaler (see page 117) object.

Description

Extracts the c-style string from the CMarshaler (see page 117) object "rMarshaler" and inserts it into the CString (see page 126) "rStr".

2.5.1.5.5 - Constructors**2.5.1.5.5.1 - CString****2.5.1.5.5.1.1 - CString::CString Constructor**

Default constructor.

C++

```
CString();
```

Description

Default constructor.

2.5.1.5.5.1.2 - CString::CString Constructor

Copy constructor.

C++

```
CString(IN const CString& rstrRhs);
```

Parameters

Parameters	Description
IN const CString& rstrRhs	CString to be used to create the new CString.

Description

Copy Constructor.

2.5.1.5.5.1.3 - CString::CString Constructor

Constructor. Constructs CString from a char buffer.

C++

```
CString(IN const char* pszRhs);
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string to be copied into the new CString buffer.

Description

Constructor.

2.5.1.5.5.1.4 - CString::CString Constructor

Constructor. Constructs CString from a char buffer of a specified length.

C++

```
CString(IN const char* pszRhs, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string to be copied into the new CString buffer.
IN unsigned int uSize	Number of characters to be copied into the new CString buffer.

Description

Constructor. Constructs CString from a char buffer of a specified length.

2.5.1.5.6 - Constructors**2.5.1.5.6.1 - CString::~CString Destructor**

Destructor.

C++

```
virtual ~CString();
```

Description

Destructor.

2.5.1.5.7 - Methods**2.5.1.5.7.1 - CString::Append Method**

Appends a c-style sub string to the end of the current CString (see page 126).

C++

```
mxt_result Append(IN const char* szSrc, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN const char* szSrc	The c-style string to append.
IN unsigned int uSize	Number of characters to append.

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

Appends "uSize" characters from the "szSrc" string at the end of the current CString (see page 126).

2.5.1.5.7.2 - CaseInsCmp**2.5.1.5.7.2.1 - CString::CaseInsCmp Method**

Case insensitive comparison between two CStrings.

C++

```
int CaseInsCmp(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	Reference to the CString (see page 126) instance with which to compare.

Returns

- 0 if equal
- < 0 if the current instance or CString (see page 126) is "smaller/shorter" than rstrCmp
- > 0 if the current instance or CString (see page 126) is "greater/longer" than rstrCmp

Description

These methods are used to compare CString (see page 126)'s internal buffer with another CString (see page 126) or a c-style string. The return values are based on the "strcmp" function.

2.5.1.5.7.2.2 - CString::CaseInsCmp Method

Case insensitive comparison between a c-style string and the current CString (see page 126).

C++

```
int CaseInsCmp(IN const char* pszCmp) const;
```

Parameters

Parameters	Description
IN const char* pszCmp	The c-style string with which to compare.

Returns

- 0 if equal
- < 0 if the current instance or CString (see page 126) is "smaller/shorter" than the compared c-style string.
- > 0 if the current instance or CString (see page 126) is "greater/longer" than the compared c-style string.

Description

See CString::CaseInsCmp(IN const CString&) (see page 131) for details.

2.5.1.5.7.3 - CString::CStr Method

Returns a pointer to the internal C-style string.

C++

```
const char* CStr() const;
```

Returns

A const pointer to the beginning of CString (see page 126)'s internal buffer.

Description

Returns a pointer to a non modifiable C-style string.

Warning

-Further calls to any method affecting CString (see page 126)'s internal buffer may cause a memory reallocation. When this reallocation occurs, any pointer returned by a preceding CStr call becomes invalid.

2.5.1.5.7.4 - Erase

2.5.1.5.7.4.1 - CString::Erase Method

Erases one character from the current CString (see page 126).

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character to be erased.

Description

Erases one character from the current CString (see page 126) object at the index "uIndex". The size value is automatically modified. The capacity is unchanged.

2.5.1.5.7.4.2 - CString::Erase Method

Erases portions of the current CString (see page 126).

C++

```
void Erase(IN unsigned int uIndex, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the first character to erase.
IN unsigned int uSize	The number of characters to erase.

Description

Erases "uSize" characters from the current CString (see page 126) object, starting at the index "uIndex". The size value is automatically modified. The capacity is unchanged.

Warning

Unlike the preceding implementation of CString (see page 126), "uSize" has no default value.

2.5.1.5.7.5 - CString::EraseAll Method

Erases all the characters from the current CString (see page 126) object.

C++

```
void EraseAll();
```

Description

A '0' character is put at the index 0 in CString (see page 126)'s internal buffer. The size is set to "0" and the capacity is unchanged.

2.5.1.5.7.6 - FindSubstring

2.5.1.5.7.6.1 - CString::FindSubstring Method

Finds the occurrence of a CString (see page 126) in the current CString (see page 126).

C++

```
unsigned int FindSubstring(IN unsigned int ustartIndex, IN const CString& rstrSubstring) const;
```

Parameters

Parameters	Description
IN unsigned int ustartIndex	The index from the current CString (see page 126) where the search begins.
IN const CString& rstrSubstring	The sub string to find.

Returns

The index of the first occurrence if it is found or the size of the string if it is not found.

Description

Starting at the index "ustartIndex", this method finds the first occurrence of "strSubstring" in the current CString (see page 126) and returns the first index where a match occurred. The comparison is case sensitive.

2.5.1.5.7.6.2 - CString::FindSubstring Method

Finds the occurrence of a c-style string in the current CString (see page 126).

C++

```
unsigned int FindSubstring(IN unsigned int ustartIndex, IN const char* pszSubstring) const;
```

Parameters

Parameters	Description
IN unsigned int ustartIndex	Index from the current CString (see page 126) where the search begins.
IN const char* pszSubstring	Sub string to find.

Returns

Index of the first occurrence if found or size of string if not found.

Description

Starting at the index "ustartIndex", this method finds the first occurrence of "pszSubstring" in the current CString (see page 126) and

returns the first index where a match occurred. The comparison is case sensitive.

2.5.1.5.7.7 - CString::Format Method

Formats arguments following pszFormat specifications.

C++

```
mxt_result Format(IN unsigned int uIndex, IN const char * pszFormat, ...);
```

2.5.1.5.7.8 - CString::FormatV Method

Formats a variable argument list following pszFormat specifications.

C++

```
mxt_result FormatV(IN unsigned int uIndex, IN const char * pszFormat, va_list args);
```

2.5.1.5.7.9 - CString::GetAt Method

Gets one character from the current CString (see page 126).

C++

```
const char& GetAt(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character to access in the string. The value must be: 0 <= uIndex <= GetSize (see page 135).

Returns

A constant reference to a constant character.

Description

Returns a reference to the character located at the index "uIndex" of the CString (see page 126).

2.5.1.5.7.10 - CString::GetBuffer Method

Returns a pointer to CString (see page 126)'s internal buffer.

C++

```
char* GetBuffer();
```

Returns

Pointer to the string "szString" from CString (see page 126)'s internal structure.

Description

Returns a pointer to CString (see page 126)'s internal buffer. Full access to CString (see page 126)'s internal buffer is granted, hence programmers must make sure not to read/write out-of-bound characters.

Warning

-Further calls to any method affecting CString (see page 126)'s internal buffer may cause a memory reallocation. When this reallocation occurs, any pointer returned by a preceding GetBuffer call becomes invalid.

2.5.1.5.7.11 - CString::GetCapacity Method

Returns the capacity of the current CString (see page 126).

C++

```
uint16_t GetCapacity() const;
```

Returns

The memory currently allocated (in bytes) to store the string.

Description

The capacity of the current Cstring's internal buffer is returned. If the current CString (see page 126) is uninitialized, GetCapacity returns "0".

2.5.1.5.7.12 - CString::GetSize Method

Returns the size of the current CString (see page 126).

C++

```
uint16_t GetSize() const;
```

Returns

The size of the current CString (see page 126) internal string (number of characters).

Description

The size of the current CString (see page 126)'s internal string is returned. If the current CString (see page 126) is uninitialized, GetSize returns "0".

2.5.1.5.7.13 - Insert**2.5.1.5.7.13.1 - CString::Insert Method**

Inserts a character into the current CString (see page 126).

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN char c);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character where the insertion takes place.
IN unsigned int uCount	The number of characters to insert.
IN char c	The character to insert.

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

Inserts "uCount" times the character "c" at the index "uIndex" of the current CString (see page 126).

2.5.1.5.7.13.2 - CString::Insert Method

Inserts a CString (see page 126) into the current CString (see page 126).

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN const CString& rstrSrc, IN unsigned int uSrcIndex, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character where the insertion takes place.
IN unsigned int uCount	The number of copies of "rstrSrc" to insert.
IN const CString& rstrSrc	The CString (see page 126) object to insert (partially or entirely) into the current CString (see page 126) object.
IN unsigned int uSrcIndex	The index of the first character of "rstrSrc" to be inserted into the current CString (see page 126).
IN unsigned int uSize	The size in bytes of the sub string to insert from "rstrSrc".

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

Inserts a sub string of "rstrSrc" at the index "uIndex" from the current CString (see page 126). This sub string starts at index "uSrcIndex" and is "uSize" characters long. The sub string is inserted "uCount" times into the current CString (see page 126) object. The total amount of characters inserted: uSize x uCount.

2.5.1.5.7.13.3 - CString::Insert Method

Inserts a c-style string into the current CString (see page 126).

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN const char* pszRhs, IN unsigned int uSize = uCOPY_ALL);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character where the insertion takes place.
IN unsigned int uCount	The number of copies of "pszRhs" to insert.
IN const char* pszRhs	The c-style string to insert into the current CString (see page 126) object. It may be terminated with NULL but this is mandatory only if uSize is equal to uCOPY_ALL.
IN unsigned int uSize = uCOPY_ALL	The length of the "pszRhs" substring.

Returns

resS_OK resFE_INVALID_ARGUMENT resFE_OUT_OF_MEMORY

Description

Inserts a sub string of "pszRhs" at the index "uIndex" from the current CString (see page 126). This sub string is "uSize" characters long. If no value is provided for "uSize", the entire string is inserted. The sub string is inserted "uCount" times into the current CString (see page 126) object. The total amount of characters inserted: uSize x uCount.

2.5.1.5.7.14 - CString::IsEmpty Method

Checks whether or not CString (see page 126)'s internal buffer is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if current CString (see page 126) object is uninitialized or has a size of 0, false otherwise.

Description

Checks whether or not the internal string from the current CString (see page 126) object has a size equal to "0".

2.5.1.5.7.15 - CString::ReduceCapacity Method

Reduces the capacity of CString (see page 126)'s internal buffer.

C++

```
mxt_result ReduceCapacity(IN uint16_t uDownToCapacity);
```

Parameters

Parameters	Description
IN uint16_t uDownToCapacity	New capacity for CString (see page 126)'s internal buffer.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Makes sure the internal buffer capacity is reduced down to the specified size. The CString (see page 126) is truncated if its size is greater than the reduced capacity value. The method call has no effect if the current capacity is lower than "uDownToCapacity".

Warning

The capacity is reduced by steps of uCSTRING_BLOCK_LENGTH bytes. Thus, a call to ReduceCapacity can sometimes have no effect

on the actual string buffer. The new capacity is always \geq to uDownToCapacity.

2.5.1.5.7.16 - CString::ReserveCapacity Method

Augments the capacity of CString (see page 126)'s internal buffer.

C++

```
mxt_result ReserveCapacity(IN uint16_t uUpToCapacity);
```

Parameters

Parameters	Description
IN uint16_t uUpToCapacity	The new capacity for CString (see page 126)'s internal buffer.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Augments the internal buffer capacity to at least the specified value. The method call has no effect if the current capacity is greater than "uUpToCapacity".

Warning

The capacity is incremented by steps of uCSTRING_BLOCK_LENGTH bytes. Thus, a call to ReserveCapacity can sometimes Reserve more capacity than required. The new capacity is always \geq to uUpToCapacity.

2.5.1.5.7.17 - CString::Resize Method

Resizes the CString (see page 126).

C++

```
void Resize(IN unsigned int uSize);
```

Parameters

Parameters	Description
IN unsigned int uSize	The new size value.

Description

Changes the size of CString (see page 126)'s internal c-style string. Further calls to GetSize (see page 135) return the new size value. A '0' is inserted at the index "uSize".

Warning

- Unlike the preceding implementation, a call to Resize with a greater size than the actual size does NOT set characters from the index "actual size" to the index "new size". Thus, a programmer should NOT assume that valid characters are present in this portion of the string.
- A call to Resize with a value for "uSize" greater than the current buffer capacity fails.

2.5.1.5.7.18 - CString::SetAt Method

Sets one character from the current CString (see page 126).

C++

```
void SetAt(IN unsigned int uIndex, IN char c);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character to be modified. The value must be: $0 \leq uIndex < \text{GetSize}$ (see page 135)
IN char c	New character value.

Description

Changes the value of the character at the index "uIndex" to a new value designated by the parameter "c".

2.5.1.5.7.19 - CString::ToLowerCase Method

Converts to lowercase the contents of the current CString (see page 126).

C++

```
void ToLowerCase();
```

Description

All characters of CString (see page 126)'s internal buffer are converted to lowercase.

Warning

This method converts characters until a '0' is encountered. Any character in the CString (see page 126) buffer beyond this point is not converted.

2.5.1.5.7.20 - CString::ToUpperCase Method

Converts to uppercase the contents of the current CString (see page 126).

C++

```
void ToUpperCase();
```

Description

All characters of CString (see page 126)'s internal buffer are converted to uppercase.

Warning

This method converts characters until a '0' is encountered. Any character in the CString (see page 126) buffer beyond this point is not converted.

2.5.1.5.7.21 - CString::TrimBothSide Method

Removes all the leading and trailing characters of the specified type.

C++

```
void TrimBothSide(IN char c = ' ');
```

Parameters

Parameters	Description
IN char c = ' '	The type of character to trim.

Description

Removes all the leading AND trailing "c" characters from the string. (default is " "). The size value is automatically modified.

2.5.1.5.7.22 - CString::TrimLeftSide Method

Removes all the leading characters of the specified type.

C++

```
void TrimLeftSide(IN char c = ' ');
```

Parameters

Parameters	Description
IN char c = ' '	The type of character to trim.

Description

Removes all the leading "c" characters from the string (default is " "). The size value is automatically modified.

2.5.1.5.7.23 - CString::TrimRightSide Method

Removes all the trailing characters of the specified type.

C++

```
void TrimRightSide(IN char c = ' ');
```

Parameters

Parameters	Description
IN char c = ' '	The type of character to trim.

Description

Removes all the trailing "c" characters from the string (default is " "). The size value is automatically modified.

2.5.1.5.8 - Operators**2.5.1.5.8.1 - !=****2.5.1.5.8.1.1 - CString::!= Operator**

Checks whether or not two CString (see page 126) objects are different.

C++

```
bool operator !=(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the size OR content of the CString (see page 126) internal buffer differs, false otherwise.

Description

Verifies that the right-hand CString (see page 126) is different than the left-hand CString (see page 126). Only the internal string of both objects are compared (the capacity and ref count values can be different).

Warning

The comparison is case sensitive.

2.5.1.5.8.1.2 - CString::!= Operator

Checks whether or not a CString (see page 126) and a c-style string are different.

C++

```
bool operator !=(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the size OR content of the current CString (see page 126) object differs from "pszRhs", false otherwise.

Description

Verifies that the right-hand c-style string is different than the left-hand CString (see page 126).

Warning

The comparison is case sensitive.

2.5.1.5.8.2 - CString::[] Operator

Direct Access operator (read only).

C++

```
const char& operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the character to access in the string.

Returns

A constant reference to a constant character.

Description

Returns a reference to the character located at the index "uIndex" of the CString (see page 126). The character returned is not modifiable, instead use the SetAtmethod to modify a single character from the current CString (see page 126).

Warning

In the previous implementation of the CString (see page 126) class, the operator[] returned a modifiable character reference.

2.5.1.5.8.3 - +**2.5.1.5.8.3.1 - CString::+ Operator**

Appends the CString (see page 126)'s content to the end of the current CString (see page 126).

C++

```
CString operator +(IN const CString& strRhs) const;
```

Parameters

Parameters	Description
IN const CString& strRhs	Right-hand CString (see page 126) operand.

Returns

A CString (see page 126) that is the result of the sum of both operands.

Description

Appends the right-hand CString (see page 126) to the end of the left-hand CString (see page 126).

2.5.1.5.8.3.2 - CString::+ Operator

Appends a character to the end of the current CString (see page 126).

C++

```
CString operator +(IN const char c) const;
```

Parameters

Parameters	Description
IN const char c	Right-hand character.

Returns

A CString (see page 126) that is the result of the sum of both operands.

Description

See CString::operator+(IN const CString&) const for details.

See Also

CString::operator+(IN const CString&) const (see page 140)

2.5.1.5.8.3.3 - CString::+ Operator

Appends a c-style string's content to the end of the current CString (see page 126).

C++

```
CString operator +(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	Right-hand c-style string.

Returns

A CString (see page 126) that is the result of the sum of both operands.

Description

See CString::operator+(IN const CString&) const for details.

See Also

CString::operator+(IN const CString&) const (see page 140)

2.5.1.5.8.4 - +=**2.5.1.5.8.4.1 - CString::+= Operator**

Appends a character to the end of the current CString (see page 126).

C++

```
CString& operator +=(IN char c);
```

Parameters

Parameters	Description
IN char c	character to append.

Returns

A reference to this CString (see page 126) instance.

Description

Appends the right-hand character to the end of the left-hand CString (see page 126). A reference to the modified CString (see page 126) object is returned.

2.5.1.5.8.4.2 - CString::+= Operator

Appends a CString (see page 126) to the end of the current CString (see page 126).

C++

```
CString& operator +=(IN const CString& rstrRhs);
```

Parameters

Parameters	Description
IN const CString& rstrRhs	CString (see page 126) to append.

Returns

A reference to this CString (see page 126) instance.

Description

Appends the right-hand CString (see page 126) to the end of the left-hand CString (see page 126). A reference to the modified CString (see page 126) object is returned.

2.5.1.5.8.4.3 - CString::+= Operator

Appends a character to the end of the current CString (see page 126).

C++

```
CString& operator +=(IN const char* pszRhs);
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string to append.

Returns

A reference to this CString (see page 126) instance.

Description

Appends the right-hand c-style string to the end of the left-hand CString (see page 126). A reference to the modified CString (see page 126) object is returned.

2.5.1.5.8.5 - <

2.5.1.5.8.5.1 - CString::< Operator

Checks whether or not the left-hand CString (see page 126) is lower than the right-hand CString (see page 126).

C++

```
bool operator <(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically less than the right-hand operand.

Description

Verifies that a CString (see page 126) is less than another CString (see page 126). The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.5.2 - CString::< Operator

Checks whether or not the left-hand CString (see page 126) is lower than the right-hand c-style string.

C++

```
bool operator <(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically less than the right-hand operand.

Description

Verifies that a CString (see page 126) is less than a c-style string. The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.6 - <=

2.5.1.5.8.6.1 - CString::<= Operator

Checks whether or not the left-hand CString (see page 126) is less than or equal to the right-hand CString (see page 126).

C++

```
bool operator <=(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically less than or equal to the right-hand operand.

Description

Verifies that a CString (see page 126) is less than or equal to another CString (see page 126). The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.6.2 - CString::<= Operator

Checks whether or not the left-hand CString (see page 126) is less than or equal to the right-hand c-style string.

C++

```
bool operator <=(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically less than or equal to the right-hand operand.

Description

Verifies that a CString (see page 126) is less than or equal to a c-style string. The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.7 - =

2.5.1.5.8.7.1 - CString::= Operator

Assigns a CString (see page 126) to the current CString (see page 126) object.

C++

```
CString& operator =(IN const CString& rstrRhs);
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object that is at the right hand side of the assignation.

Returns

A reference to this CString (see page 126) instance.

Description

Assignment operators are used to assign a CString (see page 126), a c-style string, or a character to this CString (see page 126) instance. Memory allocation differs depending on the MXD_STRING_DISABLE_REFCOUNT (see page 307) value. When the ref count scheme is enabled, a pointer copy to the internal structure is made and the RefCount is increased. When the ref count scheme is disabled, a copy of all data members (i.e. a deep copy) is assigned to this CString (see page 126) instance.

See Also

[MxD_STRING_DISABLE_REFCOUNT](#) (see page 307).

2.5.1.5.8.7.2 - CString::= Operator

Assigns a character to the current CString (see page 126) object.

C++

```
CString& operator =(IN const char c);
```

Parameters

Parameters	Description
IN const char c	The character that is at the right-hand side of the assignation.

Returns

A reference to this CString (see page 126) instance.

Description

See operator=(IN const CString (see page 126)& rstrRhs) for details.

2.5.1.5.8.7.3 - CString::= Operator

Assigns a c-style string to the current CString (see page 126) object.

C++

```
CString& operator =(IN const char* pszRhs);
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string object that is at the right-hand side of the assignation.

Returns

A reference to this CString (see page 126) instance.

Description

See operator=(IN const CString (see page 126)& rstrRhs) for details.

Note that if the pszRhs length is greater than uCSTRING_MAX_VALID_SIZE, the destination string remains unchanged, to be consistent with the Insert (see page 135) methods.

See Also

Insert (see page 135)

2.5.1.5.8.8 - ==**2.5.1.5.8.8.1 - CString::== Operator**

Checks whether or not two CString (see page 126) objects are identical.

C++

```
bool operator ==(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object that is at the right-hand side of the comparison.

Returns

True if the size and content of both CString (see page 126) objects are identical, false otherwise.

Description

Verifies that the right-hand CString (see page 126) is identical to the left-hand CString (see page 126). Only the internal string content of both objects are compared (the capacity and ref count values can be different).

Warning

The comparison is case sensitive. An uninitialized CString (see page 126) and a CString (see page 126) of size = 0 are considered identical.

2.5.1.5.8.8.2 - CString::== Operator

If a c-string object is identical with a c-style string.

C++

```
bool operator ==(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string that is at the right-hand side of the comparison.

Returns

True if the size and content of both strings are identical, false otherwise.

Description

Verifies that the right-hand CString (see page 126) is identical to the left-hand CString (see page 126).

Warning

The comparison is case sensitive. An uninitialized CString (see page 126), a CString (see page 126) of size = 0, and a NULL c-style string are considered identical.

2.5.1.5.8.9 - >**2.5.1.5.8.9.1 - CString::> Operator**

Checks whether or not the left-hand CString (see page 126) is greater than the right-hand CString (see page 126).

C++

```
bool operator >(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically greater than the right-hand operand.

Description

Verifies that a CString (see page 126) is greater than another CString (see page 126). Comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.9.2 - CString::> Operator

Checks whether or not the left-hand CString (see page 126) is greater than the right-hand c-style string.

C++

```
bool operator >(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically greater than the right-hand operand.

Description

Verifies that a CString (see page 126) is higher than a c-style string. The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.10 - >=**2.5.1.5.8.10.1 - CString::>= Operator**

Checks whether or not the left-hand CString (see page 126) is greater than or equal to the right-hand CString (see page 126).

C++

```
bool operator >=(IN const CString& rstrRhs) const;
```

Parameters

Parameters	Description
IN const CString& rstrRhs	The CString (see page 126) object at the right-hand side of the comparison.

Returns

True if the left-hand operand is lexically greater than or equal to the right-hand operand.

Description

Verifies that a CString (see page 126) is greater than or equal to another CString (see page 126). The comparison is case sensitive and based on the "strcmp" function.

2.5.1.5.8.10.2 - CString::>= Operator

Checks whether or not the left-hand CString (see page 126) is greater than or equal to the right-hand c-style string.

C++

```
bool operator >=(IN const char* pszRhs) const;
```

Parameters

Parameters	Description
IN const char* pszRhs	The c-style string at the right-hand side of the comparison.

Returns

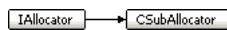
True if the left-hand operand is lexically greater than or equal to the right-hand operand.

Description

Verifies that a CString (see page 126) is greater than or equal to a c-style string. The comparison is case sensitive and based on the "strcmp" function.

2.5.1.6 - CSubAllocator Class

This class performs sub-allocation in a huge memory block allocated upon creation of the class.

Class Hierarchy**C++**

```
class CSubAllocator : public IAllocator;
```

Description

This class performs sub-allocation in a large memory block allocated upon creation of the class. This is true when MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is defined, when it is not defined, this class is only a wrapper over MX_NEW_ARRAY (see page 510)/MX_DELETE_ARRAY (see page 510).

Sub-allocation when MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT is defined:

Upon creation, this class allocates a big buffer (default size is 2048 bytes but it is fully customizable through the definition of MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES (see page 277)).

When the Allocate (see page 148) method is called, it returns a pointer in that memory block. When the memory block has insufficient size for the allocation, a new memory block of the same size or bigger is allocated and chained to the others. The returned pointer MUST be released through the Release (see page 149) method of the same allocator on which the Allocate (see page 148) call has been performed.

Furthermore, the pointers returned by Allocate (see page 148) are always aligned according to MXD_MINIMAL_ALIGNMENT_IN_BYTES (see page 298).

Sub-allocation when MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT is NOT defined:

When the sub-allocator is not enabled, this class becomes a simple wrapper over MX_NEW_ARRAY (see page 510) and MX_DELETE_ARRAY (see page 510). Therefore, the MXD_MINIMAL_ALIGNMENT_IN_BYTES (see page 298) constraint stated in the previous section is not applicable. In fact, when MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is NOT defined, the default alignment of the platform prevails.

Location

Cap/CSubAllocator.h

Constructors

Constructor	Description
• CSubAllocator (see page 147)	Default Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CSubAllocator (see page 148)	Destructor.

Legend

•	Method
V	virtual

Methods

Method	Description
• V Allocate (see page 148)	
• GetTotalAllocatedBytes (see page 148)	Returns the total number of allocated bytes.
• GetTotalAvailableBytes (see page 148)	Returns the total number of available bytes.
• GetTotalNumAllocations (see page 148)	Returns the total number of allocations.
• GetTotalNumReleases (see page 149)	Returns the total number of releases.
• V Release (see page 149)	Releases a list of SBlocks.
• Renew (see page 149)	Releases all allocations and restores SubAllocator to the initial state. Parameters: uMemoryBlockSizeInBytes: The new default size for the big memory blocks. If it is 0, the original default value is used.

IAllocator Class

IAllocator Class	Description
• A Allocate (see page 157)	Allocates a list of SBlocks.
• A Release (see page 157)	Releases a list of SBlocks.

Legend

•	Method
V	virtual
A	abstract

2.5.1.6.1 - Constructors

2.5.1.6.1.1 - CSubAllocator::CSubAllocator Constructor

Default Constructor.

C++

```
CSubAllocator( IN unsigned int uMemoryBlockSizeInBytes = g_uDEFAULT_MEMORY_BLOCK_IN_BYTES );
```

Description

Default constructor.

2.5.1.6.2 - Destructors

2.5.1.6.2.1 - CSubAllocator::~CSubAllocator Destructor

Destructor.

C++

```
virtual ~CSubAllocator();
```

Description

Destructor.

2.5.1.6.3 - Methods

2.5.1.6.3.1 - CSubAllocator::Allocate Method

```
virtual SBlock* GO Allocate(IN unsigned int uBlockCount, IN unsigned int uBlockSize, IN SBlock** ppstLastBlock = NULL);
```

2.5.1.6.3.2 - CSubAllocator::GetTotalAllocatedBytes Method

Returns the total number of allocated bytes.

C++

```
uint32_t GetTotalAllocatedBytes();
```

Returns

The total number of allocated bytes.

Description

Returns the total number of bytes that are managed as allocation buffers by the sub-allocator. Not all bytes are necessary yet sub-allocated. If MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is not defined, this function always returns 0.

2.5.1.6.3.3 - CSubAllocator::GetTotalAvailableBytes Method

Returns the total number of available bytes.

C++

```
uint32_t GetTotalAvailableBytes();
```

Returns

The total number of available bytes.

Description

Returns the total number bytes that are available for sub-allocation. If MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is not defined, this function always returns 0.

2.5.1.6.3.4 - CSubAllocator::GetTotalNumAllocations Method

Returns the total number of allocations.

C++

```
uint32_t GetTotalNumAllocations();
```

Returns

The total allocation number.

Description

Returns the total number of allocations. If MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is not defined, this function always returns 0.

2.5.1.6.3.5 - CSubAllocator::GetTotalNumReleases Method

Returns the total number of releases.

C++

```
uint32_t GetTotalNumReleases();
```

Returns

The total release number.

Description

Returns the total number of releases. If MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is not defined, this function always returns 0.

2.5.1.6.3.6 - CSubAllocator::Release Method

Releases a list of SBlocks.

C++

```
virtual void Release(IN TOA SBlock* pstBlock, IN unsigned int uBlockSize);
```

Parameters

Parameters	Description
pstBlock	A pointer to the list of allocated blocks that must be released.
uBlockSize	The minimum size of each block. It must be the same as the one that was previously passed to Allocate when the block was allocated.

Description

This method releases a list of SBlocks. Each block must be chained to the next by using the SBlock::pstNextBlock member. The last block of the list must be terminated by NULL. It is also possible to call Release multiple times to free a whole list of blocks if chaining them is not an option. Remember that you still need to set pstNext to NULL. The block size is provided to ease the implementation of some allocators.

2.5.1.6.3.7 - CSubAllocator::Renew Method

Releases all allocations and restores SubAllocator to the initial state.

Paramaters: uMemoryBlockSizeInBytes: The new default size for the big memory blocks. If it is 0, the original default value is used.

C++

```
void Renew(IN unsigned int uMemoryBlockSizeInBytes = 0);
```

Description

Releases all allocations and restores the object to its initial state. If MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT (see page 277) is not defined, nothing is done. If uMemoryBlockSizeInBytes is not 0, it is used as the new default size for the memory block allocations, otherwise the original default size, that was passed in the constructor, is used.

2.5.1.7 - CVersion Class

Class used for version checking.

Class Hierarchy

```
CVersion
```

C++

```
class CVersion;
```

Description

This is the class used for version checking. The version of the application can be set and modified for comparison with other CVersion objects.

Location

Cap/CVersion.h

Constructors

Constructor	Description
~CVersion (see page 150)	Basic constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CVersion (see page 151)	DO NOT set destructor virtual: sizeof (CVersion) MUST be == sizeof (uint32_t) (see page 85))

Legend

	Method
--	--------

Operators

Operator	Description
!= (see page 154)	Different than operator.
< (see page 154)	Less than operator.
<= (see page 155)	Less than or equal to operator.
= (see page 155)	Assignment operator.
== (see page 155)	Equal to operator.
> (see page 155)	Greater than operator.
>= (see page 156)	Greater than or equal to operator.

Legend

	Method
--	--------

Methods

Method	Description
GetBuild (see page 151)	Gets the build version number.
GetMajor (see page 151)	Gets the major version number.
GetMinor (see page 152)	Gets the minor version number.
GetRelease (see page 152)	Gets the release version number.
GetStr (see page 152)	Gets the version number in string format.
IsInRange (see page 152)	Boundaries included
SetBuild (see page 153)	Sets the build version number.
SetMajor (see page 153)	Sets the major version number.
SetMinor (see page 153)	Sets the minor version number.
SetRelease (see page 153)	Sets the release version number.
SetStr (see page 154)	Sets the version number in string format.

Legend

	Method
--	--------

2.5.1.7.1 - Constructors

2.5.1.7.1.1 - CVersion

2.5.1.7.1.1.1 - CVersion::CVersion Constructor

Basic constructor.

C++

```
CVersion();
```

Description

This is the basic constructor. The version number is initialized to 0.0.0.0.

2.5.1.7.1.1.2 - CVersion::CVersion Constructor

Copy constructor.

C++

```
CVersion(IN const CVersion& rVersion);
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the CVersion object to copy.

Description

Copy constructor.

2.5.1.7.1.1.3 - CVersion::CVersion Constructor

Constructor.

C++

```
CVersion(IN const char* pszVersion);
```

Parameters

Parameters	Description
IN const char* pszVersion	Pointer to a string containing the version number.

Description

Constructor that takes a string representation of the version number and converts it.

2.5.1.7.2 - Destructors

2.5.1.7.2.1 - CVersion::~CVersion Destructor

DO NOT set destructor virtual: sizeof (CVersion) (see page 149) MUST be == sizeof (uint32_t) (see page 85)

C++

```
~CVersion();
```

Description

Destructor.

2.5.1.7.3 - Methods

2.5.1.7.3.1 - CVersion::GetBuild Method

Gets the build version number.

C++

```
uint8_t GetBuild() const;
```

Returns

Returns the build version number.

Description

Gets the build version number stored in the CVersion (see page 149) object.

2.5.1.7.3.2 - CVersion::GetMajor Method

Gets the major version number.

C++

```
uint8_t GetMajor() const;
```

Returns

Returns the major version number.

Description

Accessors

Gets the major version number stored in the CVersion (see page 149) object.

2.5.1.7.3.3 - CVersion::GetMinor Method

Gets the minor version number.

C++

```
uint8_t GetMinor() const;
```

Returns

Returns the minor version number.

Description

Gets the minor version number stored in the CVersion (see page 149) object.

2.5.1.7.3.4 - CVersion::GetRelease Method

Gets the release version number.

C++

```
uint8_t GetRelease() const;
```

Returns

Returns the release version number.

Description

Gets the release version number stored in the CVersion (see page 149) object.

2.5.1.7.3.5 - CVersion::GetStr Method

Gets the version number in string format.

C++

```
void GetStr(OUT char* pszVersion, IN unsigned int uCapacity) const;
```

Parameters

Parameters	Description
OUT char* pszVersion	Buffer to hold the version number in string format.
IN unsigned int uCapacity	Capacity of the pszVersion buffer.

Description

Gets the version number in string format.

2.5.1.7.3.6 - CVersion::IsInRange Method

Boundaries included

C++

```
bool IsInRange(IN CVersion rVersion1, IN CVersion rVersion2) const;
```

Parameters

Parameters	Description
IN CVersion rVersion1	Reference to the CVersion (see page 149) for which the current CVersion (see page 149) must be greater than or equal to.
IN CVersion rVersion2	Reference to the CVersion (see page 149) for which the current CVersion (see page 149) must be less than or equal to.

Returns

True if the current CVersion (see page 149) is greater than or equal to rVersion1 and less than or equal to rVersion2.

Description

Verifies whether or not the current CVersion (see page 149) is greater than or equal to rVersion1 and less than or equal to rVersion2.

2.5.1.7.3.7 - CVersion::SetBuild Method

Sets the build version number.

C++

```
void SetBuild(IN uint8_t unBuild);
```

Parameters

Parameters	Description
IN uint8_t unBuild	Build version number

Description

Sets the build version number to store in the CVersion (see page 149) object.

2.5.1.7.3.8 - CVersion::SetMajor Method

Sets the major version number.

C++

```
void SetMajor(IN uint8_t unMajor);
```

Parameters

Parameters	Description
IN uint8_t unMajor	Major version number

Description

Sets the major version number to store in the CVersion (see page 149) object.

2.5.1.7.3.9 - CVersion::SetMinor Method

Sets the minor version number.

C++

```
void SetMinor(IN uint8_t unMinor);
```

Parameters

Parameters	Description
IN uint8_t unMinor	Minor version number

Description

Sets the minor version number to store in the CVersion (see page 149) object.

2.5.1.7.3.10 - CVersion::SetRelease Method

Sets the release version number.

C++

```
void SetRelease(IN uint8_t unRelease);
```

Parameters

Parameters	Description
IN uint8_t unRelease	Release version number

Description

Sets the release version number to store in the CVersion (see page 149) object.

2.5.1.7.3.11 - CVersion::SetStr Method

Sets the version number in string format.

C++

```
bool SetStr(IN const char* pszVersion);
```

Parameters

Parameters	Description
IN const char* pszVersion	Buffer holding the version number in string format.

Returns

True if the string is converted into a version number, false otherwise.

Description

Sets the version number using the string pszVersion. The string must be in x.x.x.x format.

2.5.1.7.4 - Operators

2.5.1.7.4.1 - CVersion::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if both CVersions are different, false otherwise.

Description

Verifies whether or not both CVersion (see page 149) objects are different.

2.5.1.7.4.2 - CVersion::< Operator

Less than operator.

C++

```
bool operator <(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if the left-hand CVersion (see page 149) is less than the right-hand one, false otherwise.

Description

Verifies whether or not the left-hand CVersion (see page 149) is less than the right-hand one.

2.5.1.7.4.3 - CVersion::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if the left-hand CVersion (see page 149) is less than or equal to the right-hand one, false otherwise.

Description

Verifies whether or not the left-hand CVersion (see page 149) is less than or equal to the right-hand one.

2.5.1.7.4.4 - CVersion::= Operator

Assignment operator.

C++

```
CVersion& operator =(IN const CVersion& rVersion);
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

The assigned CVersion (see page 149).

Description

Operators

Assigns the right-hand CVersion (see page 149) object into the left-hand one.

2.5.1.7.4.5 - CVersion::== Operator

Equal to operator.

C++

```
bool operator ==(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if both CVersions are equal, false otherwise.

Description

Verifies whether or not both CVersion (see page 149) object are equal.

2.5.1.7.4.6 - CVersion::> Operator

Greater than operator.

C++

```
bool operator >(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if the left-hand CVersion (see page 149) is greater than the right-hand one, false otherwise.

Description

Verifies whether or not the left-hand CVersion (see page 149) is greater than the right-hand one.

2.5.1.7.4.7 - CVersion::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CVersion& rVersion) const;
```

Parameters

Parameters	Description
IN const CVersion& rVersion	Reference to the right-hand CVersion (see page 149).

Returns

True if the left-hand CVersion (see page 149) is greater than or equal to the right-hand one, false otherwise.

Description

Verifies whether or not the left-hand CVersion (see page 149) is greater than or equal to the right-hand one.

2.5.1.8 - IAllocator Class

Interface allowing the allocation and deallocation of free blocks.

Class Hierarchy

IAllocator

C++

```
class IAllocator;
```

Description

This interface allows the allocation and the deallocation of free blocks. It is possible to allocate and release one or more blocks with one call. To do this, free blocks must be chained and the list must be terminated with a NULL pointer.

Note to the implementers:

1- Blocks may physically be of a bigger size than what was initially required. 2- Blocks must never be packed. Each block must be accessible using aligned memory access. 3- If Allocate (see page 157) fails because of an out of memory condition, the partially allocated blocks must be freed and NULL must be returned.

Location

Cap/IAllocator.h

Methods

Method	Description
◆ A Allocate (see page 157)	Allocates a list of SBlocks.
◆ A Release (see page 157)	Releases a list of SBlocks.

Legend

◆	Method
◆ A	abstract

2.5.1.8.1 - Methods

2.5.1.8.1.1 - IAllocator::Allocate Method

Allocates a list of SBlocks.

C++

```
virtual SBlock* GO Allocate(IN unsigned int uBlockCount, IN unsigned int uBlockSize, IN SBlock** ppstLastBlock = NULL) = 0;
```

Parameters

Parameters	Description
IN unsigned int uBlockCount	The number of blocks to allocate.
IN unsigned int uBlockSize	The minimum size of each block. The allocated block size is at least sizeof(SBlock) (see page 157) in size.
IN SBlock** ppstLastBlock = NULL	A pointer to a pointer to the last allocated block. It may be NULL. In such a case, the pointer to the last allocated block is not updated.

Returns

A pointer to the list of allocated blocks. NULL if the allocation fails.

Description

This method allocates a list of SBlocks. Each block is chained to the next by using the SBlock::pstNextBlock member. The last block of the list must be terminated by NULL. The block size is provided to ease the implementation of some allocators.

Some allocators do not support the block allocation of different sizes and are thus limited to one specific size.

It is important to understand that it is possible to get more allocated blocks than what was specified in the parameter. This is also to ease the implementation of some allocators.

2.5.1.8.1.2 - IAllocator::Release Method

Releases a list of SBlocks.

C++

```
virtual void Release(IN TOA SBlock* pstBlock, IN unsigned int uBlockSize) = 0;
```

Parameters

Parameters	Description
IN TOA SBlock* pstBlock	A pointer to the list of allocated blocks that must be released.
IN unsigned int uBlockSize	The minimum size of each block. It must be the same as the one that was previously passed to Allocate (see page 157) when the block was allocated.

Description

This method releases a list of SBlocks. Each block must be chained to the next by using the SBlock::pstNextBlock member. The last block of the list must be terminated by NULL. It is also possible to call Release multiple times to free a whole list of blocks if chaining them is not an option. Remember that you still need to set pstNext to NULL. The block size is provided to ease the implementation of some allocators.

2.5.2 - Structures

This section documents the structures of the Sources/Cap folder.

Structs

Struct	Description
SBlock (see page 157)	Structure used to hold a chained list of memory blocks.

2.5.2.1 - IAllocator::SBlock Struct

Structure used to hold a chained list of memory blocks.

C++

```
struct SBlock {
    SBlock* pstNextBlock;
};
```

Members

Members	Description
SBlock* <i>pstNextBlock</i> ;	Next memory block in the chained list.

2.5.3 - Templates

This section documents the templates of the Sources/Cap folder.

Templates

Template	Description
CAATree (see page 158)	Implements an AA tree, which is a type of Binary Search Tree.
CList (see page 170)	The CList class implements a double linked list of typed elements.
CMap (see page 186)	Implements an associative container that allows to retrieve values by using keys.
CMapPair (see page 196)	Class used to handle pairs of elements in map files.
CPair (see page 199)	Class providing a container to hold two distinct elements.
CPool (see page 204)	The class CPool implements a typed element pool.
CQueue (see page 208)	Interface for basic containers so they can be used as a standard queue.
CStack (see page 215)	Interface for basic containers so they are used as a standard stack.
CUncmp (see page 223)	Class allowing the use of comparison operators within container classes.
CVector (see page 227)	Class implementing a standard vector.
CVList (see page 244)	Class implementing a standard vector list.

2.5.3.1 - CAATree Template

Implements an AA tree, which is a type of Binary Search Tree.

Class Hierarchy



C++

```
template <class _Type>
class CAATree : protected CAATreeBase;
```

Description

The CAATree class provides an implementation of an AA tree, which is a type of Binary Search Tree (BST).

The AA trees are similar in many ways to the Red-Black trees. They are almost as fast as Red-Black trees but a lot simpler to implement. In this implementation, the elements are linked together by nodes holding the balancing information of the tree. Changing the linkage of a node (by inserting or deleting elements in the tree) does not change the physical address of its associated element.

The CAATree class has the following properties:

- Insert (see page 167): $O(\log n)$
- Erase (see page 163): $O(\log n)$
- Find: $O(\log n)$

It is possible to iterate over the tree using indexes. Inorder tree traversal is used and an internal iterator is kept to improve performance. Scanning a tree from its smallest to its biggest element (and vice versa) is an $O(n)$ operation.

In summary, the CAATree:

- Stores objects in an ordered way (defined by a comparison function).
- Allows fast searches among many objects.
- Does not modify the physical addresses of its elements after they are inserted.
- Does not allow insertion of duplicate objects.
- Manages its own memory buffer.
- Increases its capacity when needed.
- Reduces its capacity when requested.
- Keeps track of how many objects are currently stored.
- Calls an element's constructor automatically at construction.

- Calls an element's destructor automatically at destruction.

When to choose CAATree:

It is important to choose your container with care. Trees are useful when fast random access to a large amount of data is needed. They are also quite efficient at insertion or deletion, but they do not support duplicate values. One frequent use of trees is to implement an associative container such as a map.

CAATree, CVector (see page 227), CList (see page 170), and CVList (see page 244) offer to the programmer different complexity trade-offs and should be used accordingly. For example, a CVector (see page 227) is the type of sequence that should be used by default when data needs to be accessed randomly by an index. A CVList (see page 244) and a CList (see page 170) would also be good choices in that case, offering more performance for insertions and deletions but requiring a little more memory by element. A CAATree should be used when working with a large amount of data that needs to be kept ordered or when a fast non-sequential access to this data is needed without using an index. Being the container needing the most memory by element, other alternatives to the CAATree should be considered if memory requirements are very tight.

Choose CAATree when:

- You work with a large amount of data and you need a fast random access to them.
- Your elements need to be ordered in the container.
- It is important to avoid movement of existing contained elements when insertions/deletions take place.

Avoid CAATree when:

- Elements cannot be compared between themselves.
- You need to access each element randomly through an index.
- Memory requirements are very tight.
- Your internal data must be layout-compatible with C.

Reminders on how to use CAATree:

1. When using a CAATree of pointers, don't forget to delete the pointers before the container is destroyed. You will otherwise produce memory leaks because the CAATree only destroys the pointers and not what they point to. Before deleting a CAATree of pointers, you should always write something like this:

```
unsigned int uIndex;
unsigned int uFirstIndex = pAAATree->GetFirstIndex();
unsigned int uLastIndex = pAAATree->GetLastIndex();
for (uIndex = uFirstIndex; uIndex <= uLastIndex; uIndex++)
{
    MX_DELETE((*pAAATree)[uIndex]);
}
MX_DELETE(pAAATree);
pAAATree = NULL;
```

2. Use the method ReserveCapacity (see page 168) to avoid unnecessary reallocations. If you neglect to do so, each new insertion in a CAATree that is too small will make it grow by 1 element, added to the unavoidable overhead.
3. The default comparison function of the CAATree uses the standard operators between elements. You must supply a custom comparison function if the elements you store must be compared differently. You must also supply a new comparison function (or override the standard operators of the stored elements) if you want to search elements using only a subset of their data. Note that you have to supply a comparison function to CAATrees of pointers if you do not want them to be sorted by pointer values.
4. CAATrees do not support duplicate values. A possible solution for this issue may be to use a container of elements in each node.
5. When iterating by index and using the Erase (see page 163) function, you have to make sure that the index is valid for the whole iteration scope. Since the Erase (see page 163) function affects the size of the container, the operator[] or the GetAt (see page 164) function should be used with caution. A good practice is to use the GetSize (see page 166) function within the iteration scope such as in the following example:

```

CAATree<SomeStruct> treeSomeStruct;
unsigned int i = 0;
for (i = 0 ; i < treeSomeStruct.GetSize(); i++ )
{
    if ( treeSomeStruct[i].bToErase )
    {
        treeSomeStruct.Erase(i);
        i--;
    }
}

```

Another good practice is to iterate from the end such as in the following example:

```

CAATree<SomeStruct> treeSomeStruct;
unsigned int i = 0;
for (i = treeSomeStruct.GetSize() ; i > 0 ; i-- )
{
    if ( treeSomeStruct[i-1].bToErase )
    {
        treeSomeStruct.Erase(i-1);
    }
}

```

6. CAATree fully supports incomparable types such as structures or complex objects. However, since the != and < operators are required by the CAATree class, a user can use the CUncmp (see page 223) template helper rather than implement operators. Please refer to the CUncmp (see page 223) class documentation for more information and code examples.

Warning

This container is not thread safe.

Notes

AA trees have been developed by Mark Allen Weiss (see <http://www.cs.fiu.edu/~weiss/>).

Location

Cap/CAATree.h

See Also

CVector (see page 227), CList (see page 170), CVList (see page 244)

Constructors

Constructor	Description
CAATree (see page 161)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CAATree (see page 162)	Default constructor.

Legend

	Method
	virtual

Operators

Operator	Description
 [] (see page 169)	Gets the element at the specified index.
 = (see page 169)	Assignment operator.

Legend

	Method
---	--------

Methods

Method	Description
 Allocate (see page 162)	Allocates one element but does not construct it.
 AllocateSorted (see page 162)	Allocates a memory zone in the proper location in the tree.

◆ Erase (see page 163)	Erases one element of the tree.
◆ EraseAll (see page 163)	Erases all the elements of the tree.
◆ EraseAllocated (see page 163)	Erases an element previously allocated by Allocate (see page 162).
◆ EraseElement (see page 163)	Erases one element of the tree.
◆ EraseSorted (see page 163)	Erases an element in a sorted container.
◆ FindPtr (see page 164)	Finds an element in the tree.
◆ FindPtrSorted (see page 164)	Finds an element in a sorted container.
◆ GetAt (see page 164)	Gets the element at the specified index.
◆ GetCapacity (see page 165)	Returns the capacity.
◆ GetEndIndex (see page 165)	Gets the index of the first unused element.
◆ GetFirstIndex (see page 165)	Gets the index of the first element of the tree.
◆ GetLastIndex (see page 165)	Gets the index of the last element of the tree.
◆ GetLockCapacity (see page 166)	Returns the lock capacity count.
◆ GetMaxElementIndex (see page 166)	Gets the index of the biggest element.
◆ GetMinElementIndex (see page 166)	Gets the index of the smallest element.
◆ GetSize (see page 166)	Gets the size of the container.
◆ Insert (see page 167)	Inserts one element in the tree.
◆ InsertAllocated (see page 167)	Inserts a previously allocated element in the tree.
◆ IsEmpty (see page 167)	Indicates whether or not the container is empty.
◆ IsFull (see page 167)	Indicates whether or not the container is full.
◆ LockCapacity (see page 168)	Locks the capacity.
◆ ReduceCapacity (see page 168)	Reduces the capacity.
◆ ReserveCapacity (see page 168)	Increases the capacity.
◆ SetComparisonFunction (see page 168)	Sets the comparison function for this container.
◆ UnlockCapacity (see page 169)	Unlocks the capacity.

Legend

	Method
---	--------

2.5.3.1.1 - Constructors

2.5.3.1.1.1 - CAATree

2.5.3.1.1.1.1 - CAATree::CAATree Constructor

Default constructor.

C++

```
CAATree(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator used to allocate and free the elements.

Description

Constructor.

2.5.3.1.1.1.2 - CAATree::CAATree Constructor

Copy constructor.

C++

```
CAATree(IN const CAATree<_Type>& rAAATree);
```

Parameters

Parameters	Description
IN const CAATree<_Type>& rAAATree	A reference to the source AA tree.

Description

Copy Constructor. Uninitialized elements allocated by the Allocate (see page 162) method are not copied.

2.5.3.1.2 - **Destructors**

Default constructor.

C++

```
virtual ~CAATree();
```

Description

Destructor.

2.5.3.1.3 - **Methods**

2.5.3.1.3.1 - **CAATree::Allocate Method**

Allocates one element but does not construct it.

C++

```
GO void* Allocate();
```

Returns

A pointer on an allocated but uninitialized element. NULL is returned when the allocation fails.

Description

Allocates an element inside the tree but does not initialize it (no constructor is called). The user is then free to do whatever it wants with the returned pointer.

Once the user has initialized the element referenced by the returned pointer (by a "placement new" for example), it must call the InsertAllocated (see page 167) method to insert it in the tree. This step is necessary because it is impossible for a tree to insert an uninitialized value in its structure. The following is an example of the use of Allocate and InsertAllocated (see page 167):

```
void* pElem = AATree.Allocate();
int* pnValue = new (pElem) int;
AATree.InsertAllocated(pElem);
```

The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and, if it is not zero, the allocate fails and returns NULL.

Uninitialized elements are not accessible by indexes until they are inserted. They must also be inserted or erased before the destruction of the tree (the behaviour is otherwise undefined).

2.5.3.1.3.2 - **CAATree::AllocateSorted Method**

Allocates a memory zone in the proper location in the tree.

C++

```
mxt_result AllocateSorted(IN const _Type& rElement, OUT void** ppAllocatedZone);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to allocate.
OUT void** ppAllocatedZone	Pointer to hold the pointer to the allocated zone.

Returns

resS_OK on success, error code otherwise.

Description

This method allocates a memory zone at the right position in the tree using rElement. The allocated zone then needs to be initialized by the user. This method is useful to the map because the comparison function only compares the keys, so the second element of the pair given to this function can be left uninitialized.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.1.3.3 - CAATree::Erase Method

Erases one element of the tree.

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the element to remove from the tree.

Description

Erases one element from the tree. This method asserts if uIndex is bigger than the last index of the tree.

2.5.3.1.3.4 - CAATree::EraseAll Method

Erases all the elements of the tree.

C++

```
void EraseAll();
```

Description

Erases all the elements of the tree. It is more efficient to use EraseAll than an Erase (see page 163) for each element. This method does not erase the uninitialized elements allocated by the Allocate (see page 162) method (you must use EraseAllocated (see page 163) for these elements).

2.5.3.1.3.5 - CAATree::EraseAllocated Method

Erases an element previously allocated by Allocate (see page 162).

C++

```
void EraseAllocated(IN TOA void* pElement);
```

Parameters

Parameters	Description
IN TOA void* pElement	A pointer on an element allocated by Allocate (see page 162).

Description

Erases an element allocated by Allocate (see page 162). No destructor is called.

A crash is likely to occur if you use a pointer not allocated by Allocate (see page 162) with this method.

2.5.3.1.3.6 - CAATree::EraseElement Method

Erases one element of the tree.

C++

```
void EraseElement(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference on an element equal to the one to remove from the tree.

Description

Erases from the tree the element equal to rElement (rElement does not need to be contained in the tree). If the element to remove is not found, then this method does nothing at all.

2.5.3.1.3.7 - CAATree::EraseSorted Method

Erases an element in a sorted container.

C++

```
void EraseSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to erase.

Description

This method erases an element in a sorted container.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.1.3.8 - FindPtr**2.5.3.1.3.8.1 - CAA Tree::FindPtr Method**

Finds an element in the tree.

C++

```
const _Type* FindPtr(IN const _Type& rElement) const;
_Type* FindPtr(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to find.

Returns

A pointer on the element found, NULL otherwise.

Description

This method searches the AA tree for the given element and returns a pointer to it if it is found.

2.5.3.1.3.9 - FindPtrSorted**2.5.3.1.3.9.1 - CAA Tree::FindPtrSorted Method**

Finds an element in a sorted container.

C++

```
const _Type* FindPtrSorted(IN const _Type& rElement) const;
_Type* FindPtrSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to find.

Description

This method finds an element in a sorted container and returns a pointer to it.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.1.3.10 - GetAt**2.5.3.1.3.10.1 - CAA Tree::GetAt Method**

Gets the element at the specified index.

C++

```
_Type& GetAt(IN unsigned int uIndex);
```

```
const _Type& GetAt(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the wanted element.

Returns

The element at uIndex.

Description

Returns the element at uIndex. A crash occurs if uIndex is greater than the last index. This method sets the current index to uIndex.

2.5.3.1.3.11 - CAA::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity, which is the number of elements already allocated that may or may not be already in use.

2.5.3.1.3.12 - CAA::GetEndIndex Method

Gets the index of the first unused element.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused element.

Description

Returns the index of the first unused element (i.e., the first invalid index).

2.5.3.1.3.13 - CAA::GetFirstIndex Method

Gets the index of the first element of the tree.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first element of the tree.

Description

Returns the index of the first element of the tree. In this implementation of a tree, the first index corresponds to the smallest element according to the comparison function. Note that inorder tree traversal is used when the index is incremented.

2.5.3.1.3.14 - CAA::GetLastIndex Method

Gets the index of the last element of the tree.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last element of the tree.

Description

Returns the index of the last element of the tree. In this implementation of a tree, the last index corresponds to the biggest element according to the comparison function. Note that inorder tree traversal is used when the index is incremented.

2.5.3.1.3.15 - CAATree::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 168) is called and decreased each time UnlockCapacity (see page 169) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 168) or ReserveCapacity (see page 168) fails.

2.5.3.1.3.16 - CAATree::GetMaxElementIndex Method

Gets the index of the biggest element.

C++

```
unsigned int GetMaxElementIndex() const;
```

Returns

The index of the "biggest" element found in the container. An empty container always returns 0.

Description

Returns the index of the biggest element of the tree according to the comparison function (which corresponds to last index). This method updates the current index of the tree.

2.5.3.1.3.17 - CAATree::GetMinElementIndex Method

Gets the index of the smallest element.

C++

```
unsigned int GetMinElementIndex() const;
```

Returns

The index of the "smallest" element found in the container. An empty container always returns 0.

Description

Returns the index of the smallest element of the tree according to the comparison function (which is 0). This method updates the current index of the tree.

2.5.3.1.3.18 - CAATree::GetSize Method

Gets the size of the container.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size, which is the number of elements currently used by the tree.

2.5.3.1.3.19 - CAATree::Insert Method

Inserts one element in the tree.

C++

```
mxt_result Insert(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to insert.

Returns

resS_OK resFE_DUPLICATE resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Inserts the given element in the tree. If the element to insert is a duplicate, then this method returns an error code to indicate the problem.

The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and, if it is not zero, the insertion fails.

2.5.3.1.3.20 - CAATree::InsertAllocated Method

Inserts a previously allocated element in the tree.

C++

```
mxt_result InsertAllocated(IN TOA void* pElement);
```

Parameters

Parameters	Description
IN TOA void* pElement	A pointer on an element allocated by Allocate (see page 162).

Returns

resS_OK resFE_DUPLICATE resFE_INVALID_ARGUMENT

Description

Inserts in the tree an element previously allocated by the Allocate (see page 162) method. The element must be initialized by the user between the call to Allocate (see page 162) and the call to InsertAllocated.

```
void* pElem = AATree.Allocate();
int* pnValue = new (pElem) int;
AATree.InsertAllocated(pElem);
```

This method can fail with the return code resFE_DUPLICATE if the given element has been initialized with a value already in the tree. The EraseAllocated (see page 163) method can then be used to free the element.

A crash is likely to occur if you use a pointer not allocated by Allocate (see page 162) with this method.

2.5.3.1.3.21 - CAATree::IsEmpty Method

Indicates whether or not the container is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the size is 0; in other words, if no elements are currently in use.

2.5.3.1.3.22 - CAATree::IsFull Method

Indicates whether or not the container is full.

C++

```
bool IsFull() const;
```

Returns

True when the container is full according to its capacity.

Description

This method returns true when the size of the container equals its capacity, i.e., there is no more room to add new elements without allocating more memory.

2.5.3.1.3.23 - CAATree::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 169) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 168) or ReserveCapacity (see page 168) fails.

2.5.3.1.3.24 - CAATree::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE

Description

Reduces the capacity. The capacity is reduced if uDownToCapacity is below the current capacity. The method fails if the lock capacity count is not 0.

2.5.3.1.3.25 - CAATree::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.1.3.26 - CAATree::SetComparisonFunction Method

Sets the comparison function for this container.

C++

```
mxt_result SetComparisonFunction(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq), IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque comparison parameter.
pfnCompare	Pointer to a comparison function.

Returns

```
resS_OK resFE_INVALID_STATE
```

Description

Sets the comparison function to be used by the algorithms for this container instance. The comparison function needs to return one of the following values: < 0 if rOneElement is "smaller" than rOtherElement. > 0 if rOneElement is "bigger" than rOtherElement. 0 if rOneElement and rOtherElement are equivalent. This function MUST be called only when the tree is empty, otherwise it returns resFE_INVALID_STATE.

2.5.3.1.3.27 - CAATree::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 168) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 168) or ReserveCapacity (see page 168) fails.

2.5.3.1.4 - Operators**2.5.3.1.4.1 - []****2.5.3.1.4.1.1 - CAATree::[] Operator**

Gets the element at the specified index.

C++

```
_Type& operator [](IN unsigned int uIndex);
const _Type& operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the wanted element.

Returns

The element at uIndex.

Description

Returns the element at uIndex. A crash occurs if uIndex is greater than the last index. This method sets the current index to uIndex.

2.5.3.1.4.2 - CAATree::= Operator

Assignment operator.

C++

```
CAATree<_Type>& operator =(IN const CAATree<_Type>& rAATree);
```

Parameters

Parameters	Description
IN const CAATree<_Type>& rAATree	A reference to the source CAATree (see page 158).

Returns

A reference to this CAATree (see page 158) instance.

Description

Assignment operator.

2.5.3.2 - CList Template

The CList class implements a double linked list of typed elements.

Class Hierarchy



C++

```
template <class _Type>
class CList : protected CListBase;
```

Description

The CList class implements a double linked list of typed elements.

The CList class has the following properties:

- Insert (see page 179) O(n)
- Erase (see page 175) O(n)
- Find (see page 176) O(n) (Search/sort algorithms implemented outside of the class itself)

This template class implements a standard double linked list.

The CList class is a sequenced container that arranges elements of a given type in a series of nodes that are chained together. Each node also contains an element. Changing the linkage of a node does not change the physical address of the associated element. An internal iterator is kept to optimize the node research.

In summary, the CList:

- Offers sequential-access with an index.
- Manages its own memory buffer.
- Auto increases its capacity when needed.
- Reduces its capacity when requested.
- Keeps track of how many objects are currently stored.
- Calls element's construct automatically at construction.
- Calls element's destructor automatically at destruction.

When to choose CList:

It is important to choose your container with care. CList is designed to be efficient at insertion and deletion from the extremities. It does not support random-access. Elements are accessed sequentially.

CList, CVector (see page 227), and CVList (see page 244) offer the programmer different complexity trade-offs and should be used accordingly. CVector (see page 227) is the type of sequence that should be used by default, CVList (see page 244) should be used when there are frequent insertions and deletions from the middle of the sequences, and CList is the data structure of choice when most insertions and deletions take place at the end of the sequence.

Choose CList when:

- You need to be able to insert a new element at an arbitrary position.
- Your elements need to be ordered in the container.

- You need to access each element through its index.
- It is important to avoid movement of existing contained elements when insertions/deletions take place (see the WARNING section below).
- You want fast insertions/deletions at extremities of the container.

Avoid CList when:

- You need fast random-access with constant search time.
- Your internal data must be layout-compatible with C.

Reminders on how to use CList:

1. When using a CList of pointers, do not forget to delete the pointers before the container is destroyed. Otherwise you will produce memory leaks because the CList only destroys the pointers and not what they point to. Before deleting a CList of pointers, you should always write something like this:

```
unsigned int uIndex;
unsigned int uSize = plist->GetSize();
for (uIndex = 0; uIndex < uSize; uIndex++)
{
    MX_DELETE((*plist)[uIndex]);
}
MX_DELETE(plist);
plist = NULL;
```

2. Use the method ReserveCapacity (see page 181) to avoid unnecessary reallocations. If you neglect to do so, each new insertion on a list that is too small makes it grow by 1 element, added to the unavoidable overhead.

3. Use the method GetCapacity (see page 177) to know exactly the number of objects the list can store without reallocation.

4. Use the method GetSize (see page 179) to know the number of currently stored elements.

5. When iterating by index and using the Erase (see page 175) method, you have to make sure that the index is valid for the whole iteration scope. Since the Erase (see page 175) method affects the size of the container, the operator[] or the GetAt (see page 177) method should be used with caution. A good practice is to use the GetSize (see page 179) method within the iteration scope such as the following example:

```
CList<SomeStruct> lstSomeStruct;
unsigned int i = 0;
for ( i = 0 ; i < lstSomeStruct.GetSize(); i++ )
{
    if ( lstSomeStruct[i].bToErase )
    {
        lstSomeStruct.Erase(i);
        i--;
    }
}
```

Another good practice is to iterate from the end such as the following example:

```
CList<SomeStruct> lstSomeStruct;
unsigned int i = 0;
for ( i = lstSomeStruct.GetSize(); i > 0; i-- )
{
    if ( lstSomeStruct[i-1].bToErase )
    {
        lstSomeStruct.Erase(i-1);
    }
}
```

6. CList fully supports incomparable types such as structures or complex objects. However, the sort methods (InsertSorted (see page 180), Sort (see page 182), Find (see page 176), FindSorted (see page 176), GetMinElementIndex (see page 178), and GetMaxElementIndex (see page 178)) can not be used except if the structure or the complex object implements the != and < operators. Alternatively, you can use the CUncmp (see page 223) template helper and provide your own comparison methods.

Please refer to the CUncmp (see page 223) class documentation for more information and code examples.

Warning

This container is not thread safe.

Location

Cap/CList.h

See Also

CVector (see page 227), CList

Constructors

Constructor	Description
~CList (see page 173)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CList (see page 173)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 183)	Different than operator.
[] (see page 184)	Gets the element at position x.
< (see page 184)	Less than operator.
<= (see page 184)	Less than or equal to operator.
= (see page 184)	Assignment operator.
== (see page 185)	Comparison operator.
> (see page 185)	Greater than operator.
>= (see page 185)	Greater then or equal to.

Legend

	Method
--	--------

Methods

Method	Description
Allocate (see page 174)	Inserts one element but does not construct it.
AllocateSorted (see page 174)	Allocates a memory zone in the proper location in the list.
Append (see page 174)	Appends one element at the end index.
Erase (see page 175)	Erases one element.
EraseAll (see page 175)	Erases all elements.
EraseSorted (see page 175)	This method erases an element in a sorted container.
Find (see page 176)	Finds an element in the list.
FindPtrSorted (see page 176)	Finds an element in a sorted container.
FindSorted (see page 176)	Finds an element in a sorted list.
GetAt (see page 177)	Gets the element at position x.
GetCapacity (see page 177)	Returns the capacity.
GetEndIndex (see page 177)	Gets the end index.
GetFirstIndex (see page 177)	Gets the first index of the list.
GetLastIndex (see page 178)	Gets the last used index.
GetLockCapacity (see page 178)	Returns the local capacity count.
GetMaxElementIndex (see page 178)	Gets the "biggest" element.
GetMinElementIndex (see page 178)	Gets the "smallest" element.
GetSize (see page 179)	Gets the size.
Insert (see page 179)	Inserts the elements contained within another list.

• InsertSorted (see page 180)	Inserts one element, leaving the container sorted.
• IsEmpty (see page 180)	Checks if the list is empty.
• IsFull (see page 180)	Returns if the list is full.
• LockCapacity (see page 181)	Locks the capacity.
• Merge (see page 181)	Moves all the elements of another list to a specific index.
• ReduceCapacity (see page 181)	Reduces the capacity.
• ReserveCapacity (see page 181)	Increases the capacity.
• SetComparisonFunction (see page 182)	Sets the default comparison function for this container.
• Sort (see page 182)	Sorts the list.
• Split (see page 182)	Moves some elements from this list to another starting from a specific index.
• Swap (see page 183)	Swaps two elements.
• UnlockCapacity (see page 183)	Unlocks the capacity.

Legend

	Method
--	--------

2.5.3.2.1 - Constructors

2.5.3.2.1.1 - CList

2.5.3.2.1.1.1 - CList::CList Constructor

Constructor.

C++

```
CList(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator used to allocate and free the elements.

Description

Constructor.

2.5.3.2.1.1.2 - CList::CList Constructor

Copy Constructor.

C++

```
CList(IN const CList<_Type>& rList);
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	A reference to the source list.

Description

Copy Constructor.

2.5.3.2.2 - Destructors

2.5.3.2.2.1 - CList::~CList Destructor

Destructor.

C++

```
virtual ~CList();
```

Description

Destructor.

2.5.3.2.3 - Methods

2.5.3.2.3.1 - CList::Allocate Method

Inserts one element but does not construct it.

C++

```
void* Allocate(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the element.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Acts like an insert but does not call the constructor. The user is then free to use the pointer to do whatever it wishes. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.2 - CList::AllocateSorted Method

Allocates a memory zone in the proper location in the list.

C++

```
mxt_result AllocateSorted(IN const _Type& rElement, OUT void** ppAllocatedZone);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to allocate.
OUT void** ppAllocatedZone	Pointer to hold the pointer to the allocated zone.

Returns

resS_OK on success, error code otherwise.

Description

This method allocates a memory zone at the right position in the list by using rElement. The allocated zone then needs to be initialized by the user. This method is useful to the map because the comparison function only compares the keys, which allows to leave uninitialized the second element of the pair given to this function.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.2.3.3 - CList::Append Method

Appends one element at the end index.

C++

```
mxt_result Append(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element used to construct a new one in the CList (see page 170).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them.

See Also

Insert (see page 179)

2.5.3.2.3.4 - Erase**2.5.3.2.3.4.1 - CList::Erase Method**

Erases one element.

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the element to erase.

Description

Erases one element at position uIndex.

2.5.3.2.3.4.2 - CList::Erase Method

Erases multiple elements.

C++

```
void Erase(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the first element to erase.
IN unsigned int uCount	The number of elements to erase.

Description

Erases multiple elements, beginning with the element at a specific index.

2.5.3.2.3.5 - CList::EraseAll Method

Erases all elements.

C++

```
void EraseAll();
```

Description

Erases all elements.

2.5.3.2.3.6 - CList::EraseSorted Method

This method erases an element in a sorted container.

C++

```
void EraseSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to erase.

Description

This method erases an element in a sorted container.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.2.3.7 - CList::Find Method

Finds an element in the list.

C++

```
unsigned int Find(IN unsigned int ustartIndex, IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN unsigned int ustartIndex	Where to start the search.
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a sequential search for rElement, starting at ustartIndex, returning the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.2.3.8 - FindPtrSorted

2.5.3.2.3.8.1 - CList::FindPtrSorted Method

Finds an element in a sorted container.

C++

```
const _Type* FindPtrSorted(IN const _Type& rElement) const;
_Type* FindPtrSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to find.

Description

This method finds an element in a sorted container and returns a pointer to it.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.2.3.9 - CList::FindSorted Method

Finds an element in a sorted list.

C++

```
unsigned int FindSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a binary search for rElement on a sorted container returning the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour. The behaviour is undefined if the container has not been sorted.

2.5.3.2.3.10 - **GetAt**

2.5.3.2.3.10.1 - **CList::GetAt Method**

Gets the element at position x.

C++

```
const _Type& GetAt(IN unsigned int uIndex) const;
_Type& GetAt(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of an element in the list.
IN unsigned int uIndex	Index of an element in the list.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.2.3.11 - **CList::GetCapacity Method**

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity. The capacity is the number of elements already allocated that may or may not be already in use.

2.5.3.2.3.12 - **CList::GetEndIndex Method**

Gets the end index.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused element.

Description

Returns the index of the first unused element.

2.5.3.2.3.13 - **CList::GetFirstIndex Method**

Gets the first index of the list.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first used element.

Description

Returns the index of the first used element.

2.5.3.2.3.14 - CList::GetLastIndex Method

Gets the last used index.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last used element.

Description

Returns the index of the last used element.

2.5.3.2.3.15 - CList::GetLockCapacity Method

Returns the local capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the local capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 181) is called and decreased each time UnlockCapacity (see page 183) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 181) or ReserveCapacity (see page 181) fails.

2.5.3.2.3.16 - CList::GetMaxElementIndex Method

Gets the "biggest" element.

C++

```
unsigned int GetMaxElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "biggest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "biggest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.2.3.17 - CList::GetMinElementIndex Method

Gets the "smallest" element.

C++

```
unsigned int GetMinElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "smallest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "smallest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.2.3.18 - CList::GetSize Method

Gets the size.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size. This is the number of elements that are allocated and already in use.

2.5.3.2.3.19 - Insert**2.5.3.2.3.19.1 - CList::Insert Method**

Inserts the elements contained within another list.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN const CList& rList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN const CList& rList	A list that contains the elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts the elements contained within another list to a specified index. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.19.2 - CList::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN unsigned int uCount	The number of elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.19.3 - CList::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN const _Type& rElement);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN unsigned int uCount	The number of elements to insert.
IN const _Type& rElement	A reference to an element used to construct the elements.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them from rElement. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.20 - CList::InsertSorted Method

Inserts one element, leaving the container sorted.

C++

```
mxt_result InsertSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element used to construct the new element.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one element and constructs it from rElement, leaving the container sorted. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.2.3.21 - CList::IsEmpty Method

Checks if the list is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the size is 0; in other words, if no elements are currently in use.

2.5.3.2.3.22 - CList::IsFull Method

Returns if the list is full.

C++

```
bool IsFull() const;
```

Returns

True when the container is full according to its capacity.

Description

This method returns true when the size of the container equals its capacity, i.e., there's no more room to add new elements without allocating more memory.

2.5.3.2.3.23 - CList::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 183) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 181) or ReserveCapacity (see page 181) fails.

2.5.3.2.3.24 - CList::Merge Method

Moves all the elements of another list to a specific index.

C++

```
mxt_result Merge(IN unsigned int uIndex, INOUT CList& rList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
INOUT CList& rList	The source list.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Moves all the elements of another list to a specific index. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.25 - CList::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Reduce the capacity. The capacity is reduced to uDownToCapacity if uDownToCapacity is less than the current capacity. The method fails if the lock capacity count is not 0.

2.5.3.2.3.26 - CList::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

```
resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY
```

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.2.3.27 - CList::SetComparisonFunction Method

Sets the default comparison function for this container.

C++

```
mxt_result SetComparisonFunction(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq), IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque comparison parameter.
pfnCompare	Pointer to a comparison function.

Returns

```
resS_OK
```

Description

Sets the default comparison function to be used in Search/Sort (see page 182) algorithms for this container instance. The comparison function needs to return one of the following values: < 0 if rOneElement is "smaller" than rOtherElement. > 0 if rOneElement is "bigger" than rOtherElement. 0 if rOneElement and rOtherElement are equivalent.

2.5.3.2.3.28 - CList::Sort Method

Sorts the list.

C++

```
void Sort(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Description

Sorts the elements in the container by using the shellsort algorithm. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.2.3.29 - CList::Split Method

Moves some elements from this list to another starting from a specific index.

C++

```
mxt_result Split(IN unsigned int uIndex, OUT CList& rList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
OUT CList& rList	The destination list.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Moves some elements from this list to another starting from a specific index. The destination list is emptied first. The capacity of the target list is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.2.3.30 - CList::Swap Method

Swaps two elements.

C++

```
void Swap(IN unsigned int uFirstIndex, IN unsigned int uSecondIndex);
```

Parameters

Parameters	Description
IN unsigned int uFirstIndex	The index of the first element to swap.
IN unsigned int uSecondIndex	The index of the second element to swap.

Description

Exchanges the position of two elements.

2.5.3.2.3.31 - CList::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 181) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 181) or ReserveCapacity (see page 181) fails.

2.5.3.2.4 - Operators

2.5.3.2.4.1 - CList::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if both lists are different, otherwise returns false.

Description

Verifies that the left hand list is different than the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.2.4.2 - []

2.5.3.2.4.2.1 - CList::[] Operator

Gets the element at position x.

C++

```
const _Type& operator [](IN unsigned int uIndex) const;
_Type& operator [](IN unsigned int uIndex);
```

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.2.4.3 - CList::< Operator

Less than operator.

C++

```
bool operator <(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if the left hand list is smaller than the right hand list, otherwise returns false.

Description

Verifies if the left hand list is less than the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.2.4.4 - CList::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if the left hand list is smaller than or equal to the right hand list, otherwise returns false.

Description

Verifies if the left hand list is less than or equal to the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.2.4.5 - CList::= Operator

Assignment operator.

C++

```
CList<_Type>& operator =(IN const CList<_Type>& rList);
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to copy.

Returns

A CList (see page 170) identical to the referenced one.

Description

Creates a copy of the referenced CList (see page 170).

2.5.3.2.4.6 - CList::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if both lists are identical, otherwise returns false.

Description

Verifies that the left hand list is identical to the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.2.4.7 - CList::> Operator

Greater than operator.

C++

```
bool operator >(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if the left hand list is greater than the right hand list, otherwise returns false.

Description

Verifies if the left hand list is greater than the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.2.4.8 - CList::>= Operator

Greater then or equal to.

C++

```
bool operator >=(IN const CList<_Type>& rList) const;
```

Parameters

Parameters	Description
IN const CList<_Type>& rList	Reference to the list to compare.

Returns

Returns true if the left hand list is greater than or equal to the right hand list, otherwise returns false.

Description

Verifies if the left hand list is greater than or equal to the right hand list. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in `strcmp`.

2.5.3.3 - CMap Template

Implements an associative container that allows to retrieve values by using keys.

Class Hierarchy

CMap

C++

```
template <class _KeyType, class _ValueType, class _ContainerType = CAATree<CMapPair<_KeyType, _ValueType> > >
class CMap;
```

Description

The CMap class implements an associative container that allows to retrieve values by using keys. It is equivalent to the STL class "map".

The properties of the CMap are those of its underlying container. By default, the CMap class uses a CAATree (see page 158) to store its elements (i.e. its pairs of key-value). Hence, the default properties of the CMap are those of the CAATree (see page 158) class:

- Insert (see page 193): $O(\log n)$
- Erase (see page 190): $O(\log n)$
- Find: $O(\log n)$

It may happen that a tree is not the most appropriate structure to use inside the map. For example, it could be more efficient to use a CVector (see page 227) when memory requirements are very tight or when frequent iterations over the map are necessary. The CMap class allows to supply the container to use by a template parameter. Because the syntax to do this is a little cumbersome, the following are two examples of definitions of a CMap, one using a CAATree (see page 158) and another using a CVector (see page 227) as their underlying containers:

```
// Defines a CMap that associates an int key with a double value. The
// container used is a CAATree.
CMap<int, double> CMap1;

// Defines a CMap that associates a char key with a pointer on a CVersion.
// The container used is a CVector.
CMap<char, CVersion*, CVector<CMapPair<char, CVersion*> > > CMap2;
```

The CMap class currently supports the CAATree (see page 158), CVector (see page 227), CList (see page 170), and CVList (see page 244) containers. The CAATree (see page 158) (the default container) is the fastest one for searches and most operations. The performance is however improved for the containers that use an array (CVector (see page 227), CVList (see page 244)) by always keeping them sorted. The CList (see page 170) container is also kept sorted when used by the CMap, although that does not affect performance very much.

In summary, the CMap:

- Allows searches of elements when their key is known.
- Does not allow association of multiple elements to the same key.
- Stores objects in an ordered way (defined by a comparison function).
- Manages its own memory buffer via its internal container.
- Auto increases its capacity when needed.
- Reduces its capacity when requested.
- Keeps track of how many objects are currently stored.
- Calls the keys' and values' constructors automatically at construction.
- Calls the keys' and values' destructors automatically at destruction.

When to choose CMap:

It is important to choose your container with care. Maps are useful when you need to retrieve a value by using a key. This kind of container is often used to create dictionaries and can be thought of as an array for which the index does not need to be an integer.

Choose a CMap mainly when you need to efficiently retrieve data according to a particular key. The other advantages/disadvantages

are the ones of the underlying container.

You MUST avoid the CMap when the keys used cannot be compared between themselves or when they can be inserted twice in the container.

Reminders on how to use CMap:

1. When the keys or the values used in the CMap are pointers, do not forget to delete them before the container is destroyed. You otherwise produce memory leaks because the CMap only destroys the pointers and not to what they point. Before deleting a CMap using pointers as keys or values, you should always write something like the following:

```
if( !pMap->IsEmpty() )
{
    unsigned int uIndex;
    unsigned int uFirstIndex = pMap->GetFirstIndex();
    unsigned int uLastIndex = pMap->GetLastIndex();
    for (uIndex = uFirstIndex; uIndex <= uLastIndex; uIndex++)
    {
        // _KeyType and _ValueType are of type CVersion*
        CPair<CVersion* const, CVersion*>& rData = (*pMap)[uIndex];
        MX_DELETE(rData.GetFirst()); // Delete key
        MX_DELETE(rData.GetSecond()); // Delete value
    }
    MX_DELETE(pMap);
    pMap = NULL;
}
```

Note that even if the operator[] returns a CPair (see page 199)<const _KeyType, _ValueType>&, the const of the first template parameter of the CPair (see page 199) must be placed after the CVersion (see page 149)* expression. The reason for this is that the template mechanism in C++ does not do a simple text substitution in this case. It rather applies the const to '_KeyType', yielding a const pointer to a CVersion (see page 149) in this example.

2. Use the method ReserveCapacity (see page 194) to avoid unnecessary reallocations. If you neglect to do so, each new insertion in a CMap that is too small makes it grow by 1 element, added to the unavoidable overhead.
3. The default comparison function of the CMap uses the standard operators between keys. You must override these operators for the type of key or supply a custom comparison function if you want them to be compared differently.
4. The CMap class does not support duplicate values. A possible solution for this issue may be to associate a container with each key.
5. When iterating by index and using the Erase (see page 190) method, you have to make sure that the index is valid for the whole iteration scope. Since the Erase (see page 190) method affects the size of the container, the operator[] or the GetAt (see page 191) method should be used with caution. A good practice is to use the GetSize (see page 193) method within the iteration scope such as the following example:

```
CMap<SomeStruct> mapSomeStruct;
unsigned int i = 0;
for ( i = 0 ; i < mapSomeStruct.GetSize(); i++ )
{
    if ( mapSomeStruct[i].bToDelete )
    {
        mapSomeStruct.Erase(i);
        i--;
    }
}
```

Another good practice is to iterate from the end such as the following example:

```
CMap<SomeStruct> mapSomeStruct;
unsigned int i = 0;
for ( i = mapSomeStruct.GetSize(); i > 0; i-- )
{
    if ( mapSomeStruct[i-1].bToDelete )
    {
        mapSomeStruct.Erase(i-1);
    }
}
```

6. CMap fully supports incomparable types such as structures or complex objects. However since the != and < operators are required by the CMap class, the user can use the CUncmp (see page 223) template helper rather than implement operators. Please refer to the CUncmp (see page 223) class documentation for more information and code examples.

Warning

This container is not thread safe.

Location

Cap/CMap.h

See Also

CAATree (see page 158), CVector (see page 227), CList (see page 170), CVList (see page 244)

Constructors

Constructor	Description
~CMap (see page 189)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CMap (see page 189)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
[] (see page 195)	Gets a value and its key from an index.
= (see page 196)	Assignment operator.

Legend

	Method
---	--------

Methods

Method	Description
Allocate (see page 189)	Allocates one value but does not construct it.
Erase (see page 190)	Removes a key and its associated value from the map.
EraseAll (see page 190)	Erases all keys and their values from the map.
EraseElement (see page 190)	Removes a key and its associated value from the map.
FindPtr (see page 190)	Finds a value in the map from its key.
GetAt (see page 191)	Gets a value and its key from an index.
GetCapacity (see page 191)	Returns the capacity of the container.
GetEndIndex (see page 191)	Gets the index of the first unused element.
GetFirstIndex (see page 191)	Gets the index of the first element of the map.
GetLastIndex (see page 192)	Gets the index of the last element of the map.
GetLockCapacity (see page 192)	Returns the lock capacity count.
GetMaxElementIndex (see page 192)	Gets the index of the biggest key.
GetMinElementIndex (see page 192)	Gets the index of the smallest key.
GetSize (see page 193)	Gets the size of the map.
Insert (see page 193)	Inserts a key and a default value in the map.
IsEmpty (see page 193)	Indicates if the container is empty.
IsFull (see page 194)	Indicates if the container is full.
LockCapacity (see page 194)	Locks the capacity.
ReduceCapacity (see page 194)	Reduces the capacity.
ReserveCapacity (see page 194)	Increases the capacity.
SetComparisonFunction (see page 195)	Sets the comparison function for the keys of the map.
UnlockCapacity (see page 195)	Unlocks the capacity.

Legend**2.5.3.3.1 - Constructors****2.5.3.3.1.1 - CMap****2.5.3.3.1.1.1 - CMap::CMap Constructor**

Constructor.

C++

```
CMap(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator used to allocate and free the elements.

Description

Default constructor.

2.5.3.3.1.1.2 - CMap::CMap Constructor

Copy constructor.

C++

```
CMap(IN const CMap<_KeyType, _ValueType, _ContainerType>& rMap);
```

Description

Copy constructor.

2.5.3.3.2 - Destructors**2.5.3.3.2.1 - CMap::~CMap Destructor**

Destructor.

C++

```
virtual ~CMap();
```

Description

Destructor.

2.5.3.3.3 - Methods**2.5.3.3.3.1 - CMap::Allocate Method**

Allocates one value but does not construct it.

C++

```
void* Allocate(IN const _KeyType& rKey);
```

Parameters

Parameters	Description
IN const _KeyType& rKey	The key where the value is allocated.

Returns

A pointer on an allocated but uninitialized value. NULL is returned when the allocation fails or when the key is a duplicate.

Description

Acts like an Insert (see page 193) but does not initialize the value (no constructor is called). The user is then free to do whatever it wants with the returned pointer.

The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the Allocate fails and returns NULL.

2.5.3.3.3.2 - CMap::Erase Method

Removes a key and its associated value from the map.

C++

```
void Erase(IN unsigned int uIndex);
```

Description

Erases one entry from the map (the key and its value). This method is likely to crash if uIndex is bigger the last index of the map.

2.5.3.3.3.3 - CMap::EraseAll Method

Erases all keys and their values from the map.

C++

```
void EraseAll();
```

Description

Erases all keys from the map. It is more efficient to use EraseAll than Erase (see page 190) for each key or index.

2.5.3.3.3.4 - CMap::EraseElement Method

Removes a key and its associated value from the map.

C++

```
void EraseElement(IN const _KeyType& rKey);
```

Parameters

Parameters	Description
IN const _KeyType& rKey	The key to remove from the map.

Description

Erases one element from the map (the key and its value). If the key to remove is not found, then this function does nothing at all.

2.5.3.3.3.5 - FindPtr**2.5.3.3.3.5.1 - CMap::FindPtr Method**

Finds a value in the map from its key.

C++

```
const _ValueType* FindPtr(IN const _KeyType& rKey) const;
_ValueType* FindPtr(IN const _KeyType& rKey, IN bool bInsertIfNotFound = false);
```

Parameters

Parameters	Description
IN const _KeyType& rKey	The key of the value to find.
IN bool bInsertIfNotFound = false	Tells to insert an empty object if key is not found (Optional on non-const CMap (see page 186))

Returns

A pointer on the value found, NULL otherwise.

Description

This function searches the map for the given key and returns a pointer to its associated value if it is found.

For a non-const CMap (see page 186), when the key is not found, it is possible to automatically perform an insertion according to bInsertIfNotFound parameter, i.e., when bInsertIfNotFound is false, NULL is returned, otherwise, an association (key, empty object) is inserted in the map and the address of the empty object is returned.

2.5.3.3.3.6 - GetAt

2.5.3.3.3.6.1 - CMap::GetAt Method

Gets a value and its key from an index.

C++

```
CPair<const _KeyType, _ValueType>& GetAt(IN unsigned int uIndex);
const CPair<const _KeyType, _ValueType>& GetAt(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The desired index.

Returns

A reference on a pair containing the key and the value at uIndex.

Description

Returns the key and the value at uIndex. A crash is likely to occur if uIndex is greater than the last index.

2.5.3.3.3.7 - CMap::GetCapacity Method

Returns the capacity of the container.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity of the container. The capacity is the number of elements already allocated that may or may not be already in use.

2.5.3.3.3.8 - CMap::GetEndIndex Method

Gets the index of the first unused element.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused element.

Description

Returns the index of the first unused element (i.e., the first invalid index).

2.5.3.3.3.9 - CMap::GetFirstIndex Method

Gets the index of the first element of the map.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first element of the map.

Description

Returns the index of the first element of the map.

2.5.3.3.3.10 - CMap::GetLastIndex Method

Gets the index of the last element of the map.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last element of the map.

Description

Returns the index of the last element of the map.

2.5.3.3.3.11 - CMap::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 194) is called and decreased each time UnlockCapacity (see page 195) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 194) or ReserveCapacity (see page 194) fails.

2.5.3.3.3.12 - CMap::GetMaxElementIndex Method

Gets the index of the biggest key.

C++

```
unsigned int GetMaxElementIndex() const;
```

Returns

The index of the "biggest" key found in the container. An empty container always returns 0.

Description

Returns the index of the biggest key of the map according to the comparison function.

2.5.3.3.3.13 - CMap::GetMinElementIndex Method

Gets the index of the smallest key.

C++

```
unsigned int GetMinElementIndex() const;
```

Returns

The index of the "smallest" key found in the container. An empty container always returns 0.

Description

Returns the index of the smallest key of the map according to the comparison function.

2.5.3.3.3.14 - CMap::GetSize Method

Gets the size of the map.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size. This is the number of elements that are allocated and already in use.

2.5.3.3.3.15 - Insert

2.5.3.3.3.15.1 - CMap::Insert Method

Inserts a key and a default value in the map.

C++

```
mxt_result Insert(IN const _KeyType& rKey);
```

Parameters

Parameters	Description
IN const _KeyType& rKey	The key to insert.

Returns

resS_OK resFE_DUPLICATE resFE_INVALID_STATE resFE_OUT_OF_MEMORY resFE_FAIL

Description

Inserts a key in the map and leaves its associated value to its default state (i.e., the default constructor is called for it). If the key to insert is already present in the map, then this method returns an error code to indicate the problem.

The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.3.3.15.2 - CMap::Insert Method

Inserts a key and its value in the map.

C++

```
mxt_result Insert(IN const _KeyType& rKey, IN const _ValueType& rValue);
```

Parameters

Parameters	Description
IN const _KeyType& rKey	The key to insert.
IN const _ValueType& rValue	The value associated with the key to insert.

Returns

resS_OK resFE_DUPLICATE resFE_INVALID_STATE resFE_OUT_OF_MEMORY resFE_FAIL

Description

Inserts the given key and value in the map. If the key to insert is a duplicate, then this method returns an error code to indicate the problem.

The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.3.3.16 - CMap::IsEmpty Method

Indicates if the container is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the size is 0; in other words, if no elements are currently in use.

2.5.3.3.3.17 - CMap::IsFull Method

Indicates if the container is full.

C++

```
bool IsFull() const;
```

Returns

True when the container is full according to its capacity.

Description

This method returns true when the size of the container equals its capacity, i.e., there's no more room to add new elements without allocating more memory.

2.5.3.3.3.18 - CMap::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 195) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 194) or ReserveCapacity (see page 194) fails.

2.5.3.3.3.19 - CMap::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE

Description

Reduces the capacity. The capacity is reduced to uDownToCapacity if uDownToCapacity is less than the current capacity. The method fails if the lock capacity count is not 0.

2.5.3.3.3.20 - CMap::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.3.3.21 - CMap::SetComparisonFunction Method

Sets the comparison function for the keys of the map.

C++

```
mxt_result SetComparisonFunction(IN int (pfnCompare)(IN const _KeyType& rOneKey, IN const _KeyType& rOtherKey,  
IN mxt_opaque opq), IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque comparison parameter.
pfnCompare	Pointer to a comparison function.

Returns

resS_OK resFE_INVALID_STATE

Description

Sets the comparison function to be used by the algorithms for the keys of this instance of the map. The comparison function needs to return one of the following values: < 0 if rOneKey is "smaller" than rOtherKey. > 0 if rOneKey is "bigger" than rOtherKey. 0 if rOneKey and rOtherKey are equivalent.

This method must be called only when the map is empty, otherwise it returns resFE_INVALID_STATE.

2.5.3.3.3.22 - CMap::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 194) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 194) or ReserveCapacity (see page 194) fails.

2.5.3.3.4 - Operators**2.5.3.3.4.1 - []****2.5.3.3.4.1.1 - CMap::[] Operator**

Gets a value and its key from an index.

C++

```
CPair<const _KeyType, _ValueType>& operator [](IN unsigned int uIndex);  
const CPair<const _KeyType, _ValueType>& operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the wanted element.

Returns

A reference on a pair containing the key and the value at ulIndex.

Description

Returns the key and the value at ulIndex. A crash is likely to occur if ulIndex is greater than the last index.

2.5.3.3.4.2 - CMap::= Operator

Assignment operator.

C++

```
CMap<_KeyType, _ValueType, _ContainerType>& operator =(IN const CMap<_KeyType, _ValueType, _ContainerType>& rMap);
```

Parameters

Parameters	Description
IN const CMap<_KeyType, _ValueType, _ContainerType>& rMap	Right hand CMap (see page 186) to assign to this.

Returns

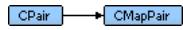
A reference to this.

Description

Assigns the right hand CMap (see page 186) to the left hand CMap (see page 186).

2.5.3.4 - CMapPair Template

Class used to handle pairs of elements in map files.

Class Hierarchy**C++**

```
template <class _Type, class _Type2>
class CMapPair : public CPair<_Type, _Type2>;
```

Description

This class is used to handle pairs of elements inside map files. These pairs usually consist of a key and some other data. It contains methods to create and compare pairs.

Location

Cap/CMap.h

See Also

CMap (see page 186), CPair (see page 199)

Constructors

Constructor	Description
~CMapPair (see page 197)	Constructor

CPair Template

CPair Template	Description
~CPair (see page 200)	Default constructor.

Legend**Destructors**

Destructor	Description
~CMapPair (see page 198)	Destructor.

CPair Template

CPair Template	Description
~CPair (see page 201)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 198)	Different than operator.
< (see page 199)	Less than operator.

CPair Template

CPair Template	Description
!= (see page 202)	Different than operator.
< (see page 202)	Less than operator.
<= (see page 202)	Less than or equal to operator.
= (see page 202)	Assignment operator.
== (see page 203)	Comparison operator.
> (see page 203)	Greater than operator.
>= (see page 203)	Greater than or equal to operator.

Legend

	Method
---	--------

Methods

Method	Description
GetFirst (see page 198)	Gets the first element.

CPair Template

CPair Template	Description
GetFirst (see page 201)	This method returns a reference to the "first" member variable.
GetSecond (see page 201)	This method returns a reference to the "second" member variable.

Legend

	Method
---	--------

2.5.3.4.1 - Constructors**2.5.3.4.1.1 - CMapPair****2.5.3.4.1.1.1 - CMapPair::CMapPair Constructor**

Constructor

C++`CMapPair();`**Description**

Default constructor.

2.5.3.4.1.1.2 - CMapPair::CMapPair Constructor

Copy constructor

C++`CMapPair(const CPair<_Type, _Type2>& rPair);`

Parameters

Parameters	Description
<code>const CPair<_Type, _Type2>& rPair</code>	The pair to copy.

Description

Copies the contents of the passed pair into this CMapPair.

2.5.3.4.1.1.3 - CMapPair::CMapPair Constructor

Constructor

C++

```
CMapPair(const _Type& rFirst, const _Type2& rSecond);
```

Parameters

Parameters	Description
<code>const _Type& rFirst</code>	First element to add to the pair.
<code>const _Type2& rSecond</code>	Second element to add to the pair.

Description

Constructor. Builds a CMapPair from the two elements.

2.5.3.4.2 - Destructors**2.5.3.4.2.1 - CMapPair::~CMapPair Destructor**

Destructor.

C++

```
virtual ~CMapPair();
```

Description

Destructor.

2.5.3.4.3 - Methods**2.5.3.4.3.1 - GetFirst****2.5.3.4.3.1.1 - CMapPair::GetFirst Method**

Gets the first element.

C++

```
const _Type& GetFirst() const;  
_Type& GetFirst();
```

Returns

A reference to the first element.

Description

Gets the first element stored in the CMapPair (see page 196).

2.5.3.4.4 - Operators**2.5.3.4.4.1 - CMapPair::!= Operator**

Different than operator.

C++

```
bool operator !=(const CMapPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CMapPair<_Type, _Type2>& rPair	Right hand pair.

Returns

True if the first element of each CMapPair (see page 196) is different.

Description

Verifies that the left hand CMapPair (see page 196) is different than the left hand one.

2.5.3.4.4.2 - CMapPair::< Operator

Less than operator.

C++

```
bool operator <(const CMapPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CMapPair<_Type, _Type2>& rPair	Right hand pair.

Returns

True if the left hand CMapPair (see page 196) is less than the right hand one.

Description

Verifies if the left hand CMapPair (see page 196) is less than the right hand one.

2.5.3.5 - CPair Template

Class providing a container to hold two distinct elements.

Class Hierarchy

CPair

C++

```
template <class _Type, class _Type2>
class CPair;
```

Description

The CPair class provides a container to hold two distinct elements. It is meant to be used by a higher level container or algorithm to bind these elements. An example would be to use the a CPair object to hold a key and an element associated with that key.

Elements stored in a CPair object must at least implement the operator< and operator!=.

Warning

This container is not thread safe.

Location

Cap/CPair.h

Constructors

Constructor	Description
CPair (see page 200)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
  ~CPair (see page 201)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 202)	Different than operator.
 < (see page 202)	Less than operator.
 <= (see page 202)	Less than or equal to operator.
 = (see page 202)	Assignment operator.
 == (see page 203)	Comparison operator.
 > (see page 203)	Greater than operator.
 >= (see page 203)	Greater than or equal to operator.

Legend

	Method
---	--------

Methods

Method	Description
 GetFirst (see page 201)	This method returns a reference to the "first" member variable.
 GetSecond (see page 201)	This method returns a reference to the "second" member variable.

Legend

	Method
--	--------

2.5.3.5.1 - Constructors

2.5.3.5.1.1 - CPair

2.5.3.5.1.1.1 - CPair::CPair Constructor

Default constructor.

C++

```
CPair();
```

Description

Constructor.

2.5.3.5.1.1.2 - CPair::CPair Constructor

Copy constructor.

C++

```
CPair(const CPair<_Type, _Type2>& rPair);
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	A reference to the source pair.

Description

Copy Constructor.

2.5.3.5.1.1.3 - CPair::CPair Constructor

Constructor. Pair of items is passed.

C++

```
CPair(const _Type& rFirst, const _Type2& rSecond);
```

Parameters

Parameters	Description
const _Type& rFirst	The element assigned to the "first" position of the pair.
const _Type2& rSecond	The element assigned to the "second" position of the pair.

Description

Constructor.

2.5.3.5.2 - Destructors

2.5.3.5.2.1 - CPair::~CPair Destructor

Destructor.

C++

```
virtual ~CPair();
```

Description

Destructor.

2.5.3.5.3 - Methods

2.5.3.5.3.1 - GetFirst

2.5.3.5.3.1.1 - CPair::GetFirst Method

This method returns a reference to the "first" member variable.

C++

```
const _Type& GetFirst() const;
_Type& GetFirst();
```

Returns

The first element of the pair.

Description

This method returns a reference to the "first" member variable.

2.5.3.5.3.2 - GetSecond

2.5.3.5.3.2.1 - CPair::GetSecond Method

This method returns a reference to the "second" member variable.

C++

```
const _Type2& GetSecond() const;
_Type2& GetSecond();
```

Returns

The second element of the pair.

Description

This method returns a reference to the "second" member variable.

2.5.3.5.4 - Operators

2.5.3.5.4.1 - CPair::!= Operator

Different than operator.

C++

```
bool operator !=(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is different than the other object's first element or, if they are equal, if the second element of this is different than to the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is different than another one.

2.5.3.5.4.2 - CPair::< Operator

Less than operator.

C++

```
bool operator <(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

A reference to the assigned CPair (see page 199).

Description

Assigns the right handed Cpair to the left handed one.

2.5.3.5.4.3 - CPair::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is smaller than the other object's first element or, if they are equal, if the second element of this is smaller than or equal to the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is smaller than or equal to another one.

2.5.3.5.4.4 - CPair:::= Operator

Assignment operator.

C++

```
CPair<_Type, _Type2>& operator =(const CPair<_Type, _Type2>& rPair);
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is smaller than the other object's first element or, if they are equal, if the second element of this is smaller than the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is strictly smaller than another one.

2.5.3.5.4.5 - CPair::== Operator

Comparison operator.

C++

```
bool operator ==(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is equal to the other object's first element and the second element of this is equal to the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is equal to another one.

2.5.3.5.4.6 - CPair::> Operator

Greater than operator.

C++

```
bool operator >(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is greater than the other object's first element or, if they are equal, if the second element of this is greater than the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is strictly greater than another one.

2.5.3.5.4.7 - CPair::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(const CPair<_Type, _Type2>& rPair) const;
```

Parameters

Parameters	Description
const CPair<_Type, _Type2>& rPair	Reference to the right hand CPair (see page 199).

Returns

True if the first element of this is greater than the other object's first element or, if they are equal, if the second element of this is greater than or equal to the other object's second element. Otherwise, returns false.

Description

Validates if a CPair (see page 199) object is greater than or equal to another one.

2.5.3.6 - CPool Template

The class CPool implements a typed element pool.

Class Hierarchy

CPool

C++

```
template <class _Type>
class CPool;
```

Description

The class CPool implements a typed element pool. It must be initialized before use. Support for concurrency protection is optional.

This class is meant to be used directly. It is not being inherited, although it is possible.

```
class CRtpPacket
{
    ...
};

void SomeFunction(...)
{
    // Constructs an element from the pool.
    CRtpPacket* pRtpPacket = CPool<CRtpPacket>::New();

    ...

    // Returns the element to the pool.
    CPool<CRtpPacket>::Delete(pRtpPacket);
    pRtpPacket = NULL;
}
```

Location

Cap/CPool.h

Methods

Method	Description
◆ Allocate (see page 205)	Allocates a new element without calling its constructor.
◆ Deallocate (see page 205)	Deallocates an element without calling its destructor.
◆ Delete (see page 205)	Destructs an element and returns it to the Pool.
◆ GetCapacity (see page 205)	Returns the capacity.
◆ GetLockCapacity (see page 205)	Returns the lock capacity.
◆ Initialize (see page 206)	Initializes the pool.
◆ LockCapacity (see page 206)	Locks the capacity.
◆ New (see page 206)	Allocates and constructs a new element by calling its default constructor.
◆ ReduceCapacity (see page 207)	Reduces the capacity.
◆ ReserveCapacity (see page 207)	Increases the capacity.
◆ Uninitialize (see page 207)	Uninitializes the pool.
◆ UnlockCapacity (see page 208)	Unlocks the capacity.

Legend

	Method
---	--------

2.5.3.6.1 - Methods

2.5.3.6.1.1 - CPool::Allocate Method

Allocates a new element without calling its constructor.

C++

```
static void* GO Allocate();
```

Returns

The newly allocated element.

Description

Allocates a new element without calling its constructor. The placement new operator should probably be called by using the returned element.

2.5.3.6.1.2 - CPool::Deallocate Method

Deallocates an element without calling its destructor.

C++

```
static void Deallocate(TOA IN void* pvoid);
```

Parameters

Parameters	Description
TOA IN void* pvoid	Pointer to the element to be deallocated.

Description

Deallocates an element.

2.5.3.6.1.3 - CPool::Delete Method

Destructs an element and returns it to the Pool.

C++

```
static void Delete(TOA IN _Type* pType);
```

Parameters

Parameters	Description
TOA IN _Type* pType	The element to delete.

Description

Deletes an element and returns it to the Pool.

2.5.3.6.1.4 - CPool::GetCapacity Method

Returns the capacity.

C++

```
static unsigned int GetCapacity();
```

Returns

The capacity.

Description

Returns the capacity. The capacity is the number of objects already allocated that may or may not be already in use.

2.5.3.6.1.5 - CPool::GetLockCapacity Method

Returns the lock capacity.

C++

```
static unsigned int GetLockCapacity();
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 206) is called and decreased each time UnlockCapacity (see page 208) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 207) or ReserveCapacity (see page 207) fails.

2.5.3.6.1.6 - CPool::Initialize Method

Initializes the pool.

C++

```
static void Initialize(IN bool bEnableThreadProtection = true, IN unsigned int uInitialCapacity = 0, IN bool bInitialCapacityLocked = false, IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN bool bEnableThreadProtection = true	True if this object is to have thread protection enabled.
IN unsigned int uInitialCapacity = 0	Sets the initial number of objects CPool (see page 204) can hold on start.
IN bool bInitialCapacityLocked = false	True if the capacity set is the maximal capacity allowed.
IN IAllocator* pAllocator = NULL	Pointer to an IAllocator (see page 156).

Description

Initializes the pool. This method must be called once per class that inherits from CPool (see page 204).

2.5.3.6.1.7 - CPool::LockCapacity Method

Locks the capacity.

C++

```
static void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 208) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 207) or ReserveCapacity (see page 207) fails.

2.5.3.6.1.8 - New**2.5.3.6.1.8.1 - CPool::New Method**

Allocates and constructs a new element by calling its default constructor.

C++

```
static _Type* GO New();
```

Returns

The newly allocated element.

Description

Allocates and constructs a new element by calling its default constructor. This method may return NULL if the pool is empty and the capacity is locked.

2.5.3.6.1.8.2 - CPool::New Method

Allocates and constructs a new element by calling its copy constructor.

C++

```
static _Type* GO New(IN _Type& rTypeFrom);
```

Parameters

Parameters	Description
IN _Type& rTypeFrom	A reference to the element to be copied into the new element.

Returns

The newly allocated element.

Description

Allocates and constructs a new element by calling its copy constructor. This method may return NULL if the pool is empty and the capacity is locked.

2.5.3.6.1.9 - CPool::ReduceCapacity Method

Reduces the capacity.

C++

```
static mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The requested capacity.

Returns

resS_OK resFE_INVALID_STATE

Description

Reduces the capacity. The capacity is reduced to uDownToCapacity if uDownToCapacity is less than the current capacity. If there are more objects currently in use (a new was called on them) than the requested capacity, the capacity is reduced but the objects remain in the pool. The free capacity is 0 until there are enough objects deleted from the pool. The method fails if the lock capacity count is not 0.

2.5.3.6.1.10 - CPool::ReserveCapacity Method

Increases the capacity.

C++

```
static mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The requested capacity.

Returns

resS_OK resFE_INVALID_STATE

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.6.1.11 - CPool::Uninitialize Method

Uninitializes the pool.

C++

```
static void Uninitialize();
```

Description

Uninitializes the pool. This method must be called once per class that inherits from CPool (see page 204) once the objects are no longer required.

2.5.3.6.1.12 - CPool::UnlockCapacity Method

Unlocks the capacity.

C++

```
static void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 206) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 207) or ReserveCapacity (see page 207) fails.

2.5.3.7 - CQueue Template

Interface for basic containers so they can be used as a standard queue.

Class Hierarchy

```
CQueue
```

C++

```
template <class _Type, class _Container = CList<_Type> >
class CQueue;
```

Description

This template class implements an interface to a basic container (CList (see page 170), CVList (see page 244), or CVector (see page 227)) to be used in a standard "queue" (FIFO) manner. It is provided for code readability reasons. Therefore, when the required container needs to be used as a queue, the CQueue interface should be preferred to the direct access to the basic container interface.

Memory management is left to the container.

The initialization of a CQueue based on a CList (see page 170) looks as follows:

```
CQueue<int, CList<int> > queue;
```

The initialization of a CQueue with the default container looks as follows:

```
CQueue<int> queue;
```

The signature of a method receiving a queue and an array of elements to be inserted into this queue would look as follows (see CQueue test case for details)

```
template <class _Queue, class _Type>
void TestAll(IN _Queue queue, IN _Type* pDataArray)
```

The CQueue class currently supports the CVector (see page 227), CList (see page 170), and CVList (see page 244) containers. CList (see page 170) has been chosen as the default container, since it is the container of choice for fast insertion/suppression at the container's extremities and this is precisely where the queue implementation performs its enqueue and dequeue operations on a CList (see page 170).

Warning

This class is not thread safe and has the same features and limitations as the container used to implement it. Refer to the basic container's documentation to know which one to use.

Make sure to reserve enough memory before you begin inserting elements into a queue object. This is done with a call to the ReserveCapacity (see page 213) method. Otherwise, if you build up a queue of N elements with consecutive calls such as the following:

```
queue.Enqueue(element);
```

you will cause N memory reallocations! Thus, a queue built with calls to Push without any previous memory allocation requires a lot more computation than building the same queue after a call to ReserveCapacity (see page 213).

Location

Cap/Cqueue.h

See Also

CVList (see page 244), CList (see page 170), CVector (see page 227), CStack (see page 215)

Constructors

Constructor	Description
• CQueue (see page 209)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CQueue (see page 210)	Destructor.

Legend

•	Method
V	virtual

Operators

Operator	Description
• != (see page 213)	Different from operator.
• < (see page 214)	Less than operator.
• <= (see page 214)	Less than or equal to operator.
• = (see page 214)	Assignment operator.
• == (see page 214)	Comparison operator.
• > (see page 215)	Greater than operator.
• >= (see page 215)	Greater than or equal to operator.

Legend

•	Method
---	--------

Methods

Method	Description
• Dequeue (see page 210)	Erases the element at the queue's first index.
• Enqueue (see page 210)	Inserts the element at the queue's last index.
• EraseAll (see page 211)	Clears the queue content.
• GetCapacity (see page 211)	Returns the capacity.
• GetFront (see page 211)	Retrieves the element at the queue's first index without erasing it.
• GetLockCapacity (see page 211)	Returns the lock capacity count.
• GetSize (see page 211)	Returns the size.
• IsEmpty (see page 212)	Queries if the queue is empty (queue size = 0).
• IsFull (see page 212)	Queries if the queue is full (queue size = max capacity).
• LockCapacity (see page 212)	Locks the capacity.
• ReduceCapacity (see page 212)	Reduces the capacity.
• ReserveCapacity (see page 213)	Increases the capacity.
• UnlockCapacity (see page 213)	Unlocks the capacity.

Legend

•	Method
---	--------

2.5.3.7.1 - Constructors

2.5.3.7.1.1 - CQueue

2.5.3.7.1.1.1 - CQueue::CQueue Constructor

Constructor.

C++

```
CQueue(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the individual elements.

Description

Constructor.

2.5.3.7.1.1.2 - CQueue::CQueue Constructor

Copy Constructor.

C++

```
CQueue( IN const CQueue<_Type, _Container>& rQueue );
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	A reference to the source queue.

Description

Copy Constructor.

2.5.3.7.2 - Destructors**2.5.3.7.2.1 - CQueue::~CQueue Destructor**

Destructor.

C++

```
virtual ~CQueue( );
```

Description

Destructor.

2.5.3.7.3 - Methods**2.5.3.7.3.1 - CQueue::Dequeue Method**

Erases the element at the queue's first index.

C++

```
void Dequeue( );
```

Description

Erases the element at the queue's first index.

2.5.3.7.3.2 - CQueue::Enqueue Method

Inserts the element at the queue's last index.

C++

```
mxt_result Enqueue( IN const _Type& rElement );
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to be inserted.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts the element at the queue's last index.

2.5.3.7.3.3 - CQueue::EraseAll Method

Clears the queue content.

C++

```
void EraseAll();
```

Description

Erases all elements from this queue object. The size is set to 0.

2.5.3.7.3.4 - CQueue::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity. The capacity is the number of elements already allocated that may or may not be already in use.

2.5.3.7.3.5 - GetFront**2.5.3.7.3.5.1 - CQueue::GetFront Method**

Retrieves the element at the queue's first index without erasing it.

C++

```
const _Type& GetFront() const;
_Type& GetFront();
```

Returns

The element at the queue's first index.

Description

Retrieves the element at the queue's first index without erasing it.

2.5.3.7.3.6 - CQueue::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 212) is called and decreased each time UnlockCapacity (see page 213) is called. As long as the lock capacity count is greater than zero, any call to ReduceCapacity (see page 212) or ReserveCapacity (see page 213) fails.

2.5.3.7.3.7 - CQueue::GetSize Method

Returns the size.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size. This is the number of elements that are allocated and already in use.

2.5.3.7.3.8 - CQueue::IsEmpty Method

Queries if the queue is empty (queue size = 0).

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the queue size is 0. An empty queue is a queue in which no elements are currently in use.

2.5.3.7.3.9 - CQueue::IsFull Method

Queries if the queue is full (queue size = max capacity).

C++

```
bool IsFull() const;
```

Returns

True if the size is equal to the capacity of the container.

Description

Returns true if the container used by the current queue object contains a number of elements equal to its capacity. It means that any new insertion of elements into this queue causes a memory reallocation.

2.5.3.7.3.10 - CQueue::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 213) is called. As long as the lock capacity count is greater than zero, any call to ReduceCapacity (see page 212) or ReserveCapacity (see page 213) fails.

2.5.3.7.3.11 - CQueue::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Reduces the capacity to uDownToCapacity if its value is below the current capacity. If uDownToCapacity is lower than the current queue size, the capacity is reduced down to the queue size (no elements are lost).

Warning

The method fails if the lock capacity count is not 0.

2.5.3.7.3.12 - CQueue::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

resS_OK **resFE_INVALID_STATE** **resFE_OUT_OF_MEMORY**

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity.

Warning

The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.7.3.13 - CQueue::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 212) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 212) or ReserveCapacity (see page 213) fails.

2.5.3.7.4 - Operators**2.5.3.7.4.1 - CQueue::!= Operator**

Different from operator.

C++

```
bool operator !=(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if either the size or the content of the queues are different, else returns false

Description

Verifies that the left hand queue is different from the right hand queue. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.7.4.2 - CQueue::< Operator

Less than operator.

C++

```
bool operator <(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if left hand queue is smaller than the right hand queue, else returns false.

Description

Verifies that the left hand queue is less than the right hand queue. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.7.4.3 - CQueue::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if the left hand queue is less than or equal to the right hand queue, else returns false.

Description

Verifies that the left hand queue is less than or equal to the right hand queue. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.7.4.4 - CQueue::== Operator

Assignment operator.

C++

```
CQueue<_Type, _Container>& operator ==(IN const CQueue<_Type, _Container>& rQueue);
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

A reference to this CQueue (see page 208) instance.

Description

Assignment operator.

2.5.3.7.4.5 - CQueue::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if both queues are equal in size and their internal content is also equal, else returns false.

Description

Verifies that both queues are equal. This is done by comparing each container element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.7.4.6 - CQueue::> Operator

Greater than operator.

C++

```
bool operator >(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if the left hand queue is greater than the right hand queue, else returns false.

Description

Verifies that the left hand queue is greater than the right hand queue. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.7.4.7 - CQueue::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CQueue<_Type, _Container>& rQueue) const;
```

Parameters

Parameters	Description
IN const CQueue<_Type, _Container>& rQueue	Reference to the CQueue (see page 208) object at the right hand side of the assignation.

Returns

Returns true if the left hand queue is greater than or equal to the right hand queue, else returns false.

Description

Verifies that the left hand queue is greater than or equal to the right hand queue. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8 - CStack Template

Interface for basic containers so they are used as a standard stack.

Class Hierarchy

```
CStack
```

C++

```
template <class _Type, class _Container = CVector<_Type> >
class CStack;
```

Description

This template class implements an interface to a basic container (CList (see page 170), CVList (see page 244), or CVector (see page 227)) to be used in a standard "stack" (LIFO) manner. It is provided for code readability reasons. Therefore, when the required container needs to be used as a stack, the CStack interface should be preferred to the direct access to the basic container interface.

Memory management is left to the container.

The initialization of a CStack based on a CList (see page 170) looks as follows:

```
CStack<int, CList<int> > stack;
```

The initialization of a CStack with the default container looks as follows:

```
CStack<int> stack;
```

The signature of a method receiving a stack and an array of elements to be inserted into this stack would look as follows (see the CStack test case for details)

```
template <class _Stack, class _Type>
void TestAll(IN _Stack stack, IN _Type* pDataArray)
```

The CStack class currently supports the CVector (see page 227), CList (see page 170), and CVList (see page 244) containers. CVector (see page 227) has been chosen as the default container, since it is the container of choice for fast insertion/suppression at the container's end and this is precisely where the stack implementation performs its push and pop operations on a CVector (see page 227).

Warning

This class is not thread safe and has the same features and limitations as the container used to implement it. Refer to the basic container documentation to know which one to use.

Make sure to reserve enough memory before you begin insertions of elements into a stack object. This is done with a call to the ReserveCapacity (see page 220) method. Otherwise, if you build up a stack of N elements with consecutive calls like the following:

```
stack.Push(element);
```

you will cause N memory reallocation! Thus, a stack built with calls to Push (see page 220) without any previous memory allocation requires a lot more computation than building the same stack after a call to ReserveCapacity (see page 220).

Location

Cap/CStack.h

See Also

CVList (see page 244), CList (see page 170), CVector (see page 227), CQueue (see page 208)

Constructors

Constructor	Description
• CStack (see page 217)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CStack (see page 217)	Destructor.

Legend

•	Method
▼	virtual

Operators

Operator	Description
• != (see page 221)	Different than operator.
• < (see page 221)	Less than operator.
• <= (see page 222)	Less than or equal to operator.
• = (see page 222)	Assignment operator.
• == (see page 222)	Equal to operator.
• > (see page 222)	Greater than operator.
• >= (see page 223)	Greater than or equal to operator.

Legend

	Method
---	--------

Methods

Method	Description
• EraseAll (see page 218)	Clears the stack content.
• GetCapacity (see page 218)	Returns the capacity.
• GetLockCapacity (see page 218)	Returns the lock capacity count.
• GetSize (see page 218)	Returns the size.
• GetTop (see page 219)	Retrieves the element at the top without erasing it.
• IsEmpty (see page 219)	Queries if the stack is empty (stack size = 0).
• IsFull (see page 219)	Queries if the stack is full (stack size = max capacity).
• LockCapacity (see page 219)	Locks the capacity.
• Pop (see page 220)	Erases the element at the top.
• Push (see page 220)	Inserts an element at the top.
• ReduceCapacity (see page 220)	Reduces the capacity.
• ReserveCapacity (see page 220)	Increases the capacity.
• UnlockCapacity (see page 221)	Unlocks the capacity.

Legend

	Method
---	--------

2.5.3.8.1 - Constructors

2.5.3.8.1.1 - CStack

2.5.3.8.1.1.1 - CStack::CStack Constructor

Constructor.

C++

```
CStack(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the individual elements.

Description

Constructor.

2.5.3.8.1.1.2 - CStack::CStack Constructor

Copy Constructor.

C++

```
CStack(IN const CStack<_Type, _Container>& rStack);
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	A reference to the source Stack.

Description

Copy Constructor.

2.5.3.8.2 - Destructors

2.5.3.8.2.1 - CStack::~CStack Destructor

Destructor.

C++

```
virtual ~CStack();
```

Description

Destructor.

2.5.3.8.3 - Methods

2.5.3.8.3.1 - CStack::EraseAll Method

Clears the stack content.

C++

```
void EraseAll();
```

Description

Erases all elements from this stack object. The size is set to 0.

2.5.3.8.3.2 - CStack::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity, which is the number of elements already allocated that may or may not be already in use.

2.5.3.8.3.3 - CStack::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 219) is called and decreased each time UnlockCapacity (see page 221) is called. As long as the lock capacity count is greater than zero, any call to ReduceCapacity (see page 220) or ReserveCapacity (see page 220) fails.

2.5.3.8.3.4 - CStack::GetSize Method

Returns the size.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size, which is the number of elements that are allocated and already in use.

2.5.3.8.3.5 - GetTop

2.5.3.8.3.5.1 - CStack::GetTop Method

Retrieves the element at the top without erasing it.

C++

```
_Type& GetTop();
```

Returns

The element at the top.

Description

Retrieves the element at the top without erasing it.

2.5.3.8.3.5.2 - CStack::GetTop Method

Retrieves the element at the top without erasing it.

C++

```
const _Type& GetTop() const;
```

Returns

The element at the top.

Description

Retrieves the element at the top without erasing it.

2.5.3.8.3.6 - CStack::IsEmpty Method

Queries if the stack is empty (stack size = 0).

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the stack size is 0. An empty stack is a stack in which no elements are currently in use.

2.5.3.8.3.7 - CStack::IsFull Method

Queries if the stack is full (stack size = max capacity).

C++

```
bool IsFull() const;
```

Returns

True if the size is equal to the capacity of the container.

Description

Returns true if the container used by the current stack object contains a number of elements equal to its capacity. It means that any new insertion of elements into this stack causes a memory reallocation.

2.5.3.8.3.8 - CStack::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time

UnlockCapacity (see page 221) is called. As long as the lock capacity count is greater than zero, any call to ReduceCapacity (see page 220) or ReserveCapacity (see page 220) fails.

2.5.3.8.3.9 - CStack::Pop Method

Erases the element at the top.

C++

```
void Pop();
```

Description

Erases the element at the top.

2.5.3.8.3.10 - CStack::Push Method

Inserts an element at the top.

C++

```
mxt_result Push(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to be inserted.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts an element at the top.

2.5.3.8.3.11 - CStack::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Reduces the capacity to uDownToCapacity if the value is less than the current capacity. If uDownToCapacity is lower than the current stack size, the capacity is reduced down to the stack size (no element is lost).

Warning

The method fails if the lock capacity count is not 0.

2.5.3.8.3.12 - CStack::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

```
resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY
```

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity.

Warning

The method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.8.3.13 - CStack::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 219) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 220) or ReserveCapacity (see page 220) fails.

2.5.3.8.4 - Operators**2.5.3.8.4.1 - CStack::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is different from the right hand stack, otherwise returns false.

Description

Verifies that the left hand stack is different than the right hand stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8.4.2 - CStack::< Operator

Less than operator.

C++

```
bool operator <(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is less than the right hand stack, otherwise returns false.

Description

Verifies that the left hand stack is less than the right hand stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8.4.3 - CStack::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is less than or equal to the right hand stack, otherwise returns false.

Description

Verifies that the left hand stack is less than or equal to the right hand stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8.4.4 - CStack:::= Operator

Assignment operator.

C++

```
CStack<_Type, _Container>& operator =(IN const CStack<_Type, _Container>& rStack);
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

A reference to this CStack (see page 215) instance.

Description

Assignment operator.

2.5.3.8.4.5 - CStack::== Operator

Equal to operator.

C++

```
bool operator ==(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is equal to the right hand stack, otherwise returns false.

Description

Verifies that the left hand stack is equal to the right hand stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8.4.6 - CStack::> Operator

Greater than operator.

C++

```
bool operator >(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is greater than the right hand stack, otherwise returns false.

Description

Verifies that the left hand stack is greater than the right hand Stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.8.4.7 - CStack::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CStack<_Type, _Container>& rStack) const;
```

Parameters

Parameters	Description
IN const CStack<_Type, _Container>& rStack	Reference to the CStack (see page 215) object at the right hand side of the assignation.

Returns

Returns true if the left hand stack is greater than or equal to the right hand stack, otherwise returns false.

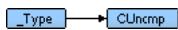
Description

Verifies that the left hand stack is greater than or equal to the right hand stack. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.9 - CUncmp Template

Class allowing the use of comparison operators within container classes.

Class Hierarchy



C++

```
template <class _Type>
class CUncmp : public _Type;
```

Description

This class is a template helper for the container classes CList (see page 170), CVList (see page 244) CVector (see page 227), CAAATree (see page 158), and CMap (see page 186). The purpose of CUncmp is to allow the use of comparison methods without having to implement the != and < operators when using incomparable types such as structures or complex objects.

CUncmp is required when the user calls one the following container class' methods:

For CList (see page 170), CVList (see page 244), and CVector (see page 227):

- InsertSorted
- Sort
- Find
- FindSorted
- GetMinElementIndex
- GetMaxElementIndex

For CAAATree (see page 158) and CMap (see page 186):

- Insert
- FindPtr

- EraseElement
- Erase

Warning

The Sort method of the container classes MUST NOT be used until the comparison function is provided, otherwise it generates an ASSERT.

Location

Cap/CUncmp.h

See Also

CList (see page 170), CVList (see page 244), CVector (see page 227), CAAATree (see page 158), CMap (see page 186)

Example

A user may want to use a CList (see page 170) template of structures and call CList::InsertSorted (see page 180) by passing its custom compare function as a parameter.

The following code shows how the user gets a compilation error when the CUncmp helper is not used:

```
#include "Config/MxConfig.h"
#include "Cap/CList.h"

typedef struct
{
    int i;
} STest;

int MyCompareFunction(IN const STest& rOneElement,
                      IN const STest& rOtherElement,
                      IN mxt_opaque opq)
{
    if (rOneElement.i < rOtherElement.i)
    {
        return -1;
    }
    else if (rOneElement.i != rOtherElement.i)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int main()
{
    CList<STest>    lstSTest;
    STest            sTest;

    // ****
    // The compiler fails because the < and != operators are not
    // defined by the STest structure. Operators are required
    // by the default internal CList::Compare method.
    // ****
    lstSTest.InsertSorted(sTest, MyCompareFunction);

    return 0;
}
```

The following code shows how the user may use the CUncmp helper to avoid compiler errors:

```
#include "Config/MxConfig.h"
#include "Cap/CList.h"
#include "Cap/CUncmp.h"

typedef struct
{
    int i;
} STest;

int MyCompareFunction(IN const CUncmp<STest>& rOneElement,
                      IN const CUncmp<STest>& rOtherElement,
                      IN mxt_opaque opq)
{
    if (rOneElement.i < rOtherElement.i)
```

```

    {
        return -1;
    }
    else if (rOneElement.i != rOtherElement.i)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int main()
{
    CList< CUncmp<STest> > lstSTest;
    STest stTest;

    lstSTest.InsertSorted(CUncmp<STest>(stTest), MyCompareFunction);

    return 0;
}

```

This helper template class inherits from the parameterized class and implements the != and < operators.

When using a CAA Tree (see page 158) or CMap (see page 186), the comparison function must be provided with the SetComparisonFunction method before adding elements to the container.

Constructors

Constructor	Description
◆ CUncmp (see page 225)	Default Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
◆ ~CUncmp (see page 226)	Destructor.

Legend

	Method
---	--------

Operators

Operator	Description
◆ != (see page 226)	Different than operator.
◆ < (see page 226)	Less than operator.

Legend

	Method
---	--------

2.5.3.9.1 - Constructors

2.5.3.9.1.1 - CUncmp

2.5.3.9.1.1.1 - CUncmp::CUncmp Constructor

Default Constructor.

C++

```
CUncmp();
```

Description

Default Constructor.

2.5.3.9.1.1.2 - CUncmp::CUncmp Constructor

Copy Constructor.

C++

```
CUncmp( IN const _Type& rType );
```

Parameters

Parameters	Description
IN const _Type& rType	A reference to _Type (parameterized class or structure).

Description

Copy Constructor.

2.5.3.9.2 - Destructors

2.5.3.9.2.1 - CUncmp::~CUncmp Destructor

Destructor.

C++

```
~CUncmp( );
```

Description

Destructor.

2.5.3.9.3 - Operators

2.5.3.9.3.1 - CUncmp::!= Operator

Different than operator.

C++

```
bool operator !=( IN const CUncmp<_Type>& rUncmp ) const;
```

Parameters

Parameters	Description
IN const CUncmp<_Type>& rUncmp	Reference to the CUncmp (see page 223) to compare.

Returns

Returns true if the address of the left hand CUncmp (see page 223) is different than the address of the right hand CUncmp (see page 223), otherwise returns false.

Description

Verifies if the left hand CUncmp (see page 223) address is different than the right hand CUncmp (see page 223) address.

This operator MUST NOT be called by the containers. This is avoided when the user provides its custom compare function to the container. Any call to this operator generates an ASSERT.

2.5.3.9.3.2 - CUncmp::< Operator

Less than operator.

C++

```
bool operator <( IN const CUncmp<_Type>& rUncmp ) const;
```

Parameters

Parameters	Description
IN const CUncmp<_Type>& rUncmp	Reference to the CUncmp (see page 223) to compare.

Returns

Returns true if the address of the left hand CUncmp (see page 223) is less than the address of the right hand CUncmp (see page 223), else returns false.

Description

Verifies if the left hand CUncmp (see page 223) address is less than the right hand CUncmp (see page 223) address.

This operator MUST NOT be called by the containers. This is avoided when the user provides its custom compare function to the container. Any call to this operator generates an ASSERT.

2.5.3.10 - CVector Template

Class implementing a standard vector.

Class Hierarchy



C++

```
template <class _Type>
class CVector : protected CVectorBase;
```

Description

This template class implements a standard vector. It is the equivalent to the STL container template class "vector" and its implementation adopts the same philosophy.

The CVector class is a sequenced container that arranges elements of a given type in a linear arrangement and allows fast random-access to any element through the use of an index (the first element is at index 0). In other words, the CVector keeps an array of elements contiguously in memory. CVectors allow constant time insertions and deletions at the end of the sequence. Inserting or deleting elements in the middle or at the start of a vector requires linear time. The internal buffer grows as elements are added to the container and when the vector is destroyed, its destructor automatically destroys the elements in the container and deallocates the memory holding those elements.

By managing its own memory, CVector eliminates the burdens of:

- Ensuring enough memory is allocated and reallocating memory when needed.
- Ensuring the memory buffer and its internal elements are correctly deleted.
- Ensuring the correct form of delete is used for single or array of elements.
- Keeping track of how many elements are stored in memory.

In summary, the CVector:

- Offers random-access with index.
- Manages its own memory buffer.
- Auto increases its capacity when needed.
- Decreases its capacity when requested.
- Keeps track of how many objects are currently stored.
- Calls the element's construct automatically at construction.
- Calls the element's destructor automatically at destruction.

More on CVector memory management:

CVector, like CString (see page 126), is a contiguous-memory container compared to CList (see page 170) that is a node-based container. As a contiguous-memory container (or if you prefer an array-based container), CVector stores its elements in one block of memory. This block of memory holds zero, one, or more elements (_Type). If a new element is inserted or an existing element is erased, the other elements in the memory block have to be shifted up or down to make room for the new element or to fill the space previously occupied by the erased element. This kind of movement affects both performance and exception safety.

As an example, suppose a CVector "vec" with its `Size() == 3` and its `Capacity() == 5` elements. As represented below, you can see that

the 3 elements are contiguous in memory.

vec ->	element1	element2	element3	free	free	
--------	----------	----------	----------	------	------	--

Now if the method "vec.Insert (see page 237)(1, 1, element4)" is called, the elements at index 2 and 3 are shifted up (to the right) and element4 is inserted in the second position.

vec ->	element1	element4	element2	element3	free	
--------	----------	----------	----------	----------	------	--

At this point in time, all **references and pointers** to element 2 and 3 become invalid.

Full reallocation is also performed when the capacity of the internal memory block is not sufficient. This reallocation always occurs when a method must increase the sequence contained in the CVector object beyond its current storage capacity.

In all such cases, references and pointers that point at altered portions of the sequence become invalid. If no reallocation occurs, only references and pointers before the insertion/deletion point remain valid.

Optimization:

Depending on the template type used, it may be possible to optimize the insertion and deletion operations. If the constructors and destructors are of no use, simply using memcpy, memmove, and memset is faster. To force this optimization, false must be passed at creation in the bUseVirtual argument.

When to choose CVector:

It is important to choose your container with care. CVector is designed to be efficient at memory management and offers random-access, but it is not always appropriate.

CList (see page 170), CVector, and CVList (see page 244) offer the programmer different complexity trade-offs and should be used accordingly. CVector is the type of sequence that should be used by default, CVList (see page 244) should be used when there are frequent insertions and deletions from the middle of the sequences, and CList (see page 170) is the data structure of choice when most insertions and deletions take place at the end of the sequence.

Choose CVector when:

- You need to be able to insert a new element at an arbitrary position.
- Your elements need to be ordered in the container.
- You need to access each element through its index.
- You need fast random-access with constant search time.
- Your internal data must be layout-compatible with C.

Avoid CVector when:

- It is important to avoid movement of existing contained elements when insertions/deletions take place (see WARNING section below).
- You want fast insertions/deletions inside the container. If frequent insertions/deletions are always at the end of the container, you can consider CVector.

Reminders on how to use CVector:

1. Avoid keeping references and pointers to internal elements. If a reference/pointer needs to be given, make sure no reallocation is performed by allocating sufficient capacity with Reserve() and by making sure no insertion or deallocation is performed on the CVector elements.
2. When using a CVector of pointers, don't forget to delete the pointers before the container is destroyed. Otherwise you will produce memory leaks because the CVector only destroys the pointers and not what they point to. Before deleting a CVector of pointers, you should always write something like this:

```
unsigned int uIndex;
unsigned int uSize = pvec->GetSize();
for (uIndex = 0; uIndex < uSize; uIndex++)
{
    MX_DELETE((*pvec)[uIndex]);
}
MX_DELETE(pvec);
```

```
pvec = NULL;
```

3. Use the method ReserveCapacity (see page 240) to avoid unnecessary reallocations. If you neglect to do so, each new insertion on a vector that is too small makes it grow by 1 element, added to the unavoidable malloc/copy/free, which is somewhat heavy.
4. Use the method GetCapacity (see page 235) to know exactly the number of objects the vector can store without reallocation.
5. Use the method GetSize (see page 237) to know the number of currently stored elements.
6. When iterating by index and using the Erase (see page 233) method, you have to make sure that the index is valid for the whole iteration scope. Since the Erase (see page 233) method affects the size of the container, the operator[] or the GetAt (see page 235) method should be used with caution. A good practice is to use the GetSize (see page 237) method within the iteration scope such as in the following example:

```
CVector<SomeStruct> vecSomeStruct;
unsigned int i = 0;
for ( i = 0 ; i < vecSomeStruct.GetSize(); i++ )
{
    if ( vecSomeStruct[i].bToErase )
    {
        vecSomeStruct.Erase(i);
        i--;
    }
}
```

Another good practice is to iterate from the end such as in the following example:

```
CVector<SomeStruct> vecSomeStruct;
unsigned int i = 0;
for ( i = vecSomeStruct.GetSize(); i > 0; i-- )
{
    if ( vecSomeStruct[i-1].bToErase )
    {
        vecSomeStruct.Erase(i-1);
    }
}
```

7. CVector fully supports incomparable types such as structures or complex objects. However, the sort methods (InsertSorted (see page 238), Sort (see page 240), Find (see page 234), FindSorted (see page 234), GetMinElementIndex (see page 236), and GetMaxElementIndex (see page 236)) can not be used except if the structure or the complex object implements the != and < operators. Alternatively, you can use the CUncmp (see page 223) template helper and provide your own compare function. Please refer to the CUncmp (see page 223) class documentation for more information and code examples.

Warning

This container is not thread safe.

Since the CVector exports methods that reorder objects, and reordering means to change the 'this' pointer of the object itself and of the aggregated objects, problems of referencing will occur if special care is not taken to update these references.

If someone however decides to store C++ objects in a CVector, and agrees to never move them since they do not support moving, it must be well documented since the potential for a problem to occur is still there. This practice is not recommended.

For C++ objects, you should normally store the object's pointer in the CVector (which yields to an access time of the same order), or use a CList (see page 170) or CVList (see page 244) when appropriate.

Location

Cap/CVector.h

See Also

CList (see page 170), CVList

Constructors

Constructor	Description
• CVector (see page 231)	Default constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CVector (see page 231)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 241)	Different than operator.
 [] (see page 242)	Gets the element at the specified index.
 < (see page 242)	Less than operator.
 <= (see page 243)	Less than or equal to operator.
 = (see page 243)	Assignment operator.
 == (see page 243)	Comparison operator.
 > (see page 243)	Greater than operator.
 >= (see page 244)	Greater than or equal to operator.

Legend

	Method
--	--------

Methods

Method	Description
 Allocate (see page 232)	Inserts one element but does not construct it.
 AllocateSorted (see page 232)	Allocates a memory zone in the proper location in the list.
 Append (see page 232)	Appends one element at the end index.
 Erase (see page 233)	Erases the element at the specified index.
 EraseAll (see page 233)	Erases all elements.
 EraseSorted (see page 233)	Erases an element in a sorted container.
 Find (see page 234)	Performs a sequential search for an element.
 FindPtrSorted (see page 234)	Finds an element in a sorted container.
 FindSorted (see page 234)	Performs a binary search for an element on a sorted container.
 GetAt (see page 235)	Gets the element at the specified index(const).
 GetCapacity (see page 235)	Returns the capacity.
 GetEndIndex (see page 235)	Gets the index of the first unused element.
 GetFirstIndex (see page 235)	Gets the first used index.
 GetLastIndex (see page 236)	Gets the last used element.
 GetLockCapacity (see page 236)	Returns the lock capacity count.
 GetMaxElementIndex (see page 236)	Performs a sequential search for the "biggest" element.
 GetMinElementIndex (see page 236)	Performs a sequential search for the "smallest" element.
 GetSize (see page 237)	Gets the size of the vector.
 Insert (see page 237)	Inserts the element contained within another vector.
 InsertSorted (see page 238)	Inserts one element, leaving the container sorted.
 IsEmpty (see page 238)	Checks if the vector is empty.
 IsFull (see page 239)	Checks if the vector is full.
 LockCapacity (see page 239)	Locks the capacity.
 Merge (see page 239)	Moves all the elements of another vector to a specific index.
 ReduceCapacity (see page 239)	Reduces the capacity.
 ReserveCapacity (see page 240)	Increases the capacity.
 SetComparisonFunction (see page 240)	Sets the default comparison function for this container.
 Sort (see page 240)	Sorts the elements in the container using the shellsort algorithm.
 Split (see page 240)	Moves some elements from this vector to another starting from a specific index.
 Swap (see page 241)	Exchanges the position of two elements.
 UnlockCapacity (see page 241)	Unlocks the capacity.

Legend

	Method
--	--------

2.5.3.10.1 - Constructors**2.5.3.10.1.1 - CVector****2.5.3.10.1.1.1 - CVector::CVector Constructor**

Default constructor.

C++

```
CVector(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the internal buffer.

Description

Constructor.

2.5.3.10.1.1.2 - CVector::CVector Constructor

Constructor.

C++

```
CVector(IN bool bUseVirtual, IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN bool bUseVirtual	True if the constructor and destructor should always be called. False if memcpy, memmove, and memset may be used instead. Set to TRUE by default.
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the internal buffer.

Description

Constructor.

2.5.3.10.1.1.3 - CVector::CVector Constructor

Copy Constructor.

C++

```
CVector(IN const CVector<_Type>& rVector);
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	A reference to the source vector.

Description

Copy Constructor.

2.5.3.10.2 - Destructors**2.5.3.10.2.1 - CVector::~CVector Destructor**

Destructor.

C++

```
virtual ~CVector();
```

Description

Destructor.

2.5.3.10.3 - Methods**2.5.3.10.3.1 - CVector::Allocate Method**

Inserts one element but does not construct it.

C++

```
void* Allocate(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the element.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Acts like an insert but does not call the constructor. The user is then free to use the pointer to do whatever it wishes. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.2 - CVector::AllocateSorted Method

Allocates a memory zone in the proper location in the list.

C++

```
mxt_result AllocateSorted(IN const _Type& rElement, OUT void** ppAllocatedZone);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to allocate.
OUT void** ppAllocatedZone	Pointer to hold the pointer to the allocated zone.

Returns

resS_OK on success, error code otherwise.

Description

This method allocates a memory zone at the right position in the vector using rElement. The allocated zone then needs to be initialized by the user. This method is useful for the map because the comparison function only compares the keys, so the second element of the pair given to this function can be left uninitialized.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.10.3.3 - CVector::Append Method

Appends one element at the end index.

C++

```
mxt_result Append(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element that is used to construct a new one in the CVector (see page 227).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Appends one element and constructs it.

See Also

Insert (see page 237)

2.5.3.10.3.4 - Erase**2.5.3.10.3.4.1 - CVector::Erase Method**

Erases the element at the specified index.

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the element to erase.

Description

Erases one element.

2.5.3.10.3.4.2 - CVector::Erase Method

Erases a number of elements at the specified index.

C++

```
void Erase(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the first element to erase.
IN unsigned int uCount	The number of elements to erase.

Description

Erases multiple elements, beginning with element at uIndex.

2.5.3.10.3.5 - CVector::EraseAll Method

Erases all elements.

C++

```
void EraseAll();
```

Description

Erases all elements.

2.5.3.10.3.6 - CVector::EraseSorted Method

Erases an element in a sorted container.

C++

```
void EraseSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to erase.

Description

This method erases an element in a sorted container.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.10.3.7 - CVector::Find Method

Performs a sequential search for an element.

C++

```
unsigned int Find(IN unsigned int ustartIndex, IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN unsigned int ustartIndex	Where to start the search.
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a sequential search for rElement, starting at ustartIndex and returns the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.10.3.8 - FindPtrSorted

2.5.3.10.3.8.1 - CVector::FindPtrSorted Method

Finds an element in a sorted container.

C++

```
const _Type* FindPtrSorted(IN const _Type& rElement) const;
_Type* FindPtrSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to find.

Description

This method finds an element in a sorted container and returns a pointer to it.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.10.3.9 - CVector::FindSorted Method

Performs a binary search for an element on a sorted container.

C++

```
unsigned int FindSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a binary search for rElement on a sorted container and returns the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour. The behaviour is undefined if the container has not been sorted.

2.5.3.10.3.10 - GetAt**2.5.3.10.3.10.1 - CVector::GetAt Method**

Gets the element at the specified index(const).

C++

```
const _Type& GetAt(IN unsigned int uIndex) const;
_Type& GetAt(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.10.3.11 - CVector::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity, which is the number of elements already allocated that may or may not be already in use.

2.5.3.10.3.12 - CVector::GetEndIndex Method

Gets the index of the first unused element.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused element.

Description

Returns the index of the first unused element.

2.5.3.10.3.13 - CVector::GetFirstIndex Method

Gets the first used index.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first used element.

Description

Returns the index of the first used element.

2.5.3.10.3.14 - CVector::GetLastIndex Method

Gets the last used element.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last used element.

Description

Returns the index of the last used element.

2.5.3.10.3.15 - CVector::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 239) is called and decreased each time UnlockCapacity (see page 241) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 239) or ReserveCapacity (see page 240) fails.

2.5.3.10.3.16 - CVector::GetMaxElementIndex Method

Performs a sequential search for the "biggest" element.

C++

```
unsigned int GetMaxElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "biggest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "biggest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.10.3.17 - CVector::GetMinElementIndex Method

Performs a sequential search for the "smallest" element.

C++

```
unsigned int GetMinElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "smallest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "smallest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.10.3.18 - CVector::GetSize Method

Gets the size of the vector.

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size, which is the number of elements that are allocated and already in use.

2.5.3.10.3.19 - Insert

2.5.3.10.3.19.1 - CVector::Insert Method

Inserts the element contained within another vector.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN const CVector& rVector);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN const CVector& rVector	A vector that contains the elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts the elements contained within another vector to a specified index. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.19.2 - CVector::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN unsigned int uCount	The number of elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.19.3 - CVector::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN const _Type& rElement);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN unsigned int uCount	The number of elements to insert.
IN const _Type& rElement	A reference to an element that is used to construct the elements.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them from rElement. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.20 - CVector::InsertSorted Method

Inserts one element, leaving the container sorted.

C++

```
mxt_result InsertSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element that is used to construct the new element.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one element and constructs it from rElement, leaving the container sorted. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.10.3.21 - CVector::IsEmpty Method

Checks if the vector is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the size is 0; in other words, if no elements are currently in use.

2.5.3.10.3.22 - CVector::IsFull Method

Checks if the vector is full.

C++

```
bool IsFull() const;
```

Returns

True when the container is full according to its capacity.

Description

This method returns true when the size of the container equals its capacity, i.e., there's no more room to add new elements without allocating more memory.

2.5.3.10.3.23 - CVector::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 241) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 239) or ReserveCapacity (see page 240) fails.

2.5.3.10.3.24 - CVector::Merge Method

Moves all the elements of another vector to a specific index.

C++

```
mxt_result Merge(IN unsigned int uIndex, INOUT CVector& rVector);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
INOUT CVector& rVector	The source vector.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Moves all the elements of another vector to uIndex. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.25 - CVector::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Reduces the capacity. The capacity is reduced to uDownToCapacity if the size of the CVector (see page 227) is below

uDownToCapacity and if uDownToCapacity is below the current capacity. This method fails if the lock capacity count is not 0.

2.5.3.10.3.26 - CVector::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. This method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.10.3.27 - CVector::SetComparisonFunction Method

Sets the default comparison function for this container.

C++

```
mxt_result SetComparisonFunction(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,  
IN mxt_opaque opq), IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque comparison parameter.
pfnCompare	Pointer to a comparison function.

Returns

resS_OK

Description

Sets the default comparison function to be used in Search/Sort (see page 240) algorithms for this container instance. The comparison function needs to return one of the following values: < 0 if rOneElement is "smaller" than rOtherElement. > 0 if rOneElement is "bigger" than rOtherElement. 0 if rOneElement and rOtherElement are equivalent.

2.5.3.10.3.28 - CVector::Sort Method

Sorts the elements in the container using the shellsort algorithm.

C++

```
void Sort(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) =  
NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Description

Sorts the elements in the container using the shellsort algorithm. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.10.3.29 - CVector::Split Method

Moves some elements from this vector to another starting from a specific index.

C++

```
mxt_result Split(IN unsigned int uIndex, OUT CVector& rVector);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
OUT CVector& rVector	The destination vector.

Returns

```
resS_OK resFE_OUT_OF_MEMORY
```

Description

Moves some elements from this vector to another starting from a specific index. The destination vector is emptied first. The capacity of the target vector is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.10.3.30 - CVector::Swap Method

Exchanges the position of two elements.

C++

```
void Swap(IN unsigned int uFirstIndex, IN unsigned int uSecondIndex);
```

Parameters

Parameters	Description
IN unsigned int uFirstIndex	The index of the first element to swap.
IN unsigned int uSecondIndex	The index of the second element to swap.

Description

Exchanges the position of two elements.

2.5.3.10.3.31 - CVector::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 239) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 239) or ReserveCapacity (see page 240) fails.

2.5.3.10.4 - Operators**2.5.3.10.4.1 - CVector::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if both vectors are different, otherwise returns false.

Description

Verifies that the left hand vector is different than the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.10.4.2 - []

2.5.3.10.4.2.1 - CVector::[] Operator

Gets the element at the specified index.

C++

```
_Type& operator [](IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.10.4.2.2 - CVector::[] Operator

Gets the element at the specified index(const).

C++

```
const _Type& operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.10.4.3 - CVector::< Operator

Less than operator.

C++

```
bool operator <(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if the left hand vector is less than the right hand vector, otherwise returns false.

Description

Verifies that the left hand vector is less than the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.10.4.4 - CVector::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if the left hand vector is less than or equal to the right hand vector, otherwise returns false.

Description

Verifies that the left hand vector is less than or equal to the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.10.4.5 - CVector:::= Operator

Assignment operator.

C++

```
CVector<_Type>& operator =(IN const CVector<_Type>& rVector);
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Reference to the assigned vector.

Description

Assigns the right hand CVector (see page 227) to the left hand one.

2.5.3.10.4.6 - CVector::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if both vectors are identical, otherwise returns false.

Description

Verifies that the left hand vector is identical to the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.10.4.7 - CVector::> Operator

Greater than operator.

C++

```
bool operator >(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if the left hand vector is greater than the right hand vector, otherwise returns false.

Description

Verifies that the left hand vector is greater than the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.10.4.8 - CVector::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CVector<_Type>& rVector) const;
```

Parameters

Parameters	Description
IN const CVector<_Type>& rVector	Reference to the list to compare.

Returns

Returns true if the left hand vector is greater than or equal to the right hand vector, otherwise returns false.

Description

Verifies that the left hand vector is greater than or equal to the right hand vector. This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11 - CVList Template

Class implementing a standard vector list.

Class Hierarchy



C++

```
template <class _Type>
class CVList : protected CVListBase;
```

Description

The CVList class is a sequenced container that manages an array of pointers to elements of the templated type and allows fast random-access to any element through the use of an index (first element is at index 0). CVList allows constant time insertions and deletions at the end of the sequence. Inserting or deleting elements in the middle or at the start of a CVList requires linear time.

By managing its own memory, CVList eliminates the burdens of:

- Ensuring enough memory is allocated and reallocating memory when needed.
- Ensuring the memory buffer and its internal elements are correctly deleted.
- Ensuring the correct form of delete is used for single or array of elements.
- Keeping track of how many elements are stored in memory.

In summary, the CVList:

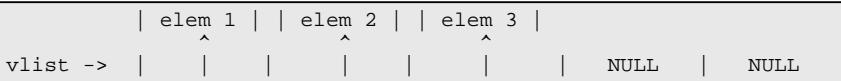
- Offers random-access with index.
- Manages its own memory.
- Auto increases its capacity when needed.
- Decreases its capacity when requested.

- Keeps track of how many objects are currently stored.
- Calls the element's construct automatically at construction.
- Calls the element's destructor automatically at destruction.

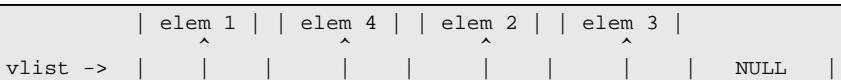
More on CVList memory management:

In a CVList, each element is kept in a separate allocation block that is sized for `_Type`. CVList manages an array of pointers to the elements. This array maintains the ordering of the elements. If a new element is inserted or an existing element is erased, other pointers to elements in the array have to be shifted up or down to make room for the new element pointer or to fill the space previously occupied by the erased element pointer. As you can understand, this kind of movement affects performance.

As an example, suppose a CVList "vlist" with its `Size() == 3` and its `Capacity() == 5` elements. As represented below, you can see that the 3 pointers to elements are contiguous in memory and that each element is allocated in an individual memory block.



Now if the method "vlist.Insert (see page 254)(1, 1, element4)" is called, the pointer to elements at index 2 and 3 are shifted up (to the right) and the pointer to element4 is inserted in the second position.



Full reallocation is also performed when the capacity of the internal memory block used to keep the pointer to the elements is not sufficient. This reallocation always occurs when a member method must increase the sequence contained in the CVList object beyond its current storage capacity.

In all such cases, references and pointers that point at elements remain valid and may be reused later.

When to choose CVList:

It is important to choose your container with care. CVList is designed to be efficient at insertion, deletion and offers random-access, but it is not always appropriate.

CLList (see page 170), CVector (see page 227), and CVList offer the programmer different complexity trade-offs and should be used accordingly. CVector (see page 227) is the type of sequence that should be used by default, CVList should be used when there are frequent insertions and deletions from the middle of the sequences, and CLList (see page 170) is the data structure of choice when most insertions and deletions take place at the end of the sequence.

Choose CVList when:

- You need to be able to insert a new element at an arbitrary position.
- Your elements need to be ordered in the container.
- You need to access each element through its index.
- You need fast random-access with constant search time.
- It is important to avoid movement of existing contained elements when insertions/deletions take place (see the WARNING section below).
- You want fast insertions/deletions inside the container. If frequent insertions/deletions are always at the end of the container, you can consider CVector (see page 227).

Avoid CVList when:

- Your internal data must be layout-compatible with C;

Reminders on how to use CVList:

1. When using a CVList of pointers, don't forget to delete the pointers before the container is destroyed. Otherwise you will produce memory leaks because the CVList only destroys the pointers and not what they point to. Before deleting a CVList of pointers, you should always write something like this:

```
unsigned int uIndex;
```

```

unsigned int uSize = pvlist->GetSize();
for (uIndex = 0; uIndex < uSize; uIndex++)
{
    MX_DELETE((*pvlist)[uIndex]);
}
MX_DELETE(pvlist);
pvlist = NULL;

```

3. Use the method ReserveCapacity (see page 256) to avoid unnecessary reallocations. If you neglect to do so, each new insertion on a CVList that is too small makes it grow by 1 element, added to the unavoidable malloc/copy/free, which is somewhat heavy.
4. Use the method GetCapacity (see page 252) to know exactly the number of objects the CVList can store without reallocation.
5. Use the method GetSize (see page 254) to know the number of elements currently stored.
6. When iterating by index and using the Erase (see page 249) method, you have to make sure that the index is valid for the whole iteration scope. Since the Erase (see page 249) method affects the size of the container, the operator[] or the GetAt (see page 251) method should be used with caution. A good practice is to use the GetSize (see page 254) method within the iteration scope such as in the following example:

```

CVList<SomeStruct> vlstSomeStruct;
unsigned int i = 0;
for (i = 0 ; i < vlstSomeStruct.GetSize(); i++ )
{
    if ( vlstSomeStruct[i].bToDelete )
    {
        vlstSomeStruct.Erase(i);
        i--;
    }
}

```

Another good practice is to iterate from the end such as in the following example:

```

CVList<SomeStruct> vlstSomeStruct;
unsigned int i = 0;
for ( i = vlstSomeStruct.GetSize(); i > 0; i-- )
{
    if ( vlstSomeStruct[i-1].bToDelete )
    {
        vlstSomeStruct.Erase(i-1);
    }
}

```

7. CVList fully supports incomparable types such as structures or complex objects. However, the sort methods (InsertSorted (see page 255), Sort (see page 257), Find (see page 250), FindSorted (see page 251), GetMinElementIndex (see page 253), and GetMaxElementIndex (see page 253)) can not be used except if the structure or the complex object implements the != and < operators. Alternatively, you can use the CUncmp (see page 223) template helper and provide your own compare function. Please refer to the CUncmp (see page 223) class documentation for more information and code examples.

Warning

This container is not thread safe.

Location

Cap/CVList.h

See Also

CList (see page 170), CVector (see page 227)

Constructors

Constructor	Description
CVList (see page 248)	Constructor.

Legend

	Constructor
	Method

Destructors

Destructor	Description
  ~CVList (see page 248)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 258)	Different than operator.
 [] (see page 259)	Returns the element at the specified index.
 < (see page 259)	Less than operator.
 <= (see page 259)	Less than or equal to operator.
 = (see page 260)	Assignment operator.
 == (see page 260)	Comparison operator.
 > (see page 260)	Greater than operator.
 >= (see page 261)	Greater than or equal to operator.

Legend

	Method
---	--------

Methods

Method	Description
 Allocate (see page 248)	Inserts one element but does not construct it.
 AllocateSorted (see page 249)	Allocates a memory zone in the proper location in the CVList.
 Append (see page 249)	Appends one element at the end index.
 Erase (see page 249)	Erases the element at the specified index.
 EraseAll (see page 250)	Erases all elements.
 EraseSorted (see page 250)	Erases an element in a sorted container.
 Find (see page 250)	Performs a sequential search for an element starting at the index.
 FindPtrSorted (see page 251)	Finds an element in a sorted container.
 FindSorted (see page 251)	Performs a binary search for an element on a sorted container.
 GetAt (see page 251)	Returns the element at the specified index.
 GetCapacity (see page 252)	Returns the capacity.
 GetEndIndex (see page 252)	Returns the index of the first unused element.
 GetFirstIndex (see page 252)	Returns the index of the first used element.
 GetLastIndex (see page 253)	Returns the index of the last used element.
 GetLockCapacity (see page 253)	Returns the lock capacity count.
 GetMaxElementIndex (see page 253)	Performs a sequential search for the "biggest" element.
 GetMinElementIndex (see page 253)	Performs a sequential search for the "smallest" element.
 GetSize (see page 254)	Gets the size of the CVList.
 Insert (see page 254)	Inserts the elements contained within another CVList.
 InsertSorted (see page 255)	Inserts one element, leaving the container sorted.
 IsEmpty (see page 255)	Checks if the CVList is empty.
 IsFull (see page 255)	Checks if the CVList is full.
 LockCapacity (see page 256)	Locks the capacity.
 Merge (see page 256)	Moves all the elements of another VList to a specific index.
 ReduceCapacity (see page 256)	Reduces the capacity.
 ReserveCapacity (see page 256)	Increases the capacity.
 SetComparisonFunction (see page 257)	Sets the default comparison function for this container.
 Sort (see page 257)	Sorts the elements in the container using the shellsort algorithm.
 Split (see page 257)	Moves some elements from this VList to another starting from a specific index.
 Swap (see page 258)	Exchanges the position of two elements.
 UnlockCapacity (see page 258)	Unlocks the capacity.

Legend

	Method
---	--------

2.5.3.11.1 - Constructors

2.5.3.11.1.1 - CVList

2.5.3.11.1.1.1 - CVList::CVList Constructor

Constructor.

C++

```
CVList(IN IAllocator* pAllocator = NULL);
```

Parameters

Parameters	Description
IN IAllocator* pAllocator = NULL	Allows to specify an allocator that is used to allocate and free the individual elements.

Description

Constructor.

2.5.3.11.1.1.2 - CVList::CVList Constructor

Copy Constructor.

C++

```
CVList(IN const CVList<_Type>& rVList);
```

Parameters

Parameters	Description
IN const CVList<_Type>& rVList	A reference to the source VList.

Description

Copy Constructor.

2.5.3.11.2 - Destructors

2.5.3.11.2.1 - CVList::~CVList Destructor

Destructor.

C++

```
virtual ~CVList();
```

Description

Destructor.

2.5.3.11.3 - Methods

2.5.3.11.3.1 - CVList::Allocate Method

Inserts one element but does not construct it.

C++

```
void* Allocate(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the element.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Acts like an insert but does not call the constructor. The user is then free to use the pointer to do whatever he wishes. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.2 - CVList::AllocateSorted Method

Allocates a memory zone in the proper location in the CVList (see page 244).

C++

```
mxt_result AllocateSorted(IN const _Type& rElement, OUT void** ppAllocatedZone);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to allocate.
OUT void** ppAllocatedZone	Pointer to hold the pointer to the allocated zone.

Returns

resS_OK on success, error code otherwise.

Description

This method allocates a memory zone at the right position in the CVList (see page 244) using rElement. The allocated zone then needs to be initialized by the user. This method is useful to the map because the comparison function only compares the keys, so the second element of the pair given to this function can be left uninitialized.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.11.3.3 - CVList::Append Method

Appends one element at the end index.

C++

```
mxt_result Append(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element that is used to construct the new one in the CVList (see page 244).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Appends one element and constructs it.

See Also

Insert (see page 254)

2.5.3.11.3.4 - Erase

2.5.3.11.3.4.1 - CVList::Erase Method

Erases the element at the specified index.

C++

```
void Erase(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the element to erase.

Description

Erases one element.

2.5.3.11.3.4.2 - CVList::Erase Method

Erases multiple elements, beginning with the element at the index.

C++

```
void Erase(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the first element to erase.
IN unsigned int uCount	The number of elements to erase.

Description

Erases multiple elements, beginning with the element at uIndex.

2.5.3.11.3.5 - CVList::EraseAll Method

Erases all elements.

C++

```
void EraseAll();
```

Description

Erases all elements.

2.5.3.11.3.6 - CVList::EraseSorted Method

Erases an element in a sorted container.

C++

```
void EraseSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to erase.

Description

This method erases an element in a sorted container.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.11.3.7 - CVList::Find Method

Performs a sequential search for an element starting at the index.

C++

```
unsigned int Find(IN unsigned int ustartIndex, IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN unsigned int ustartIndex	Where to start the search.
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a sequential search for rElement, starting at ustartIndex, returning the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.11.3.8 - FindPtrSorted**2.5.3.11.3.8.1 - CVList::FindPtrSorted Method**

Finds an element in a sorted container.

C++

```
const _Type* FindPtrSorted(IN const _Type& rElement) const;
_Type* FindPtrSorted(IN const _Type& rElement);
```

Parameters

Parameters	Description
IN const _Type& rElement	Reference to the element to find.

Description

This method finds an element in a sorted container and returns a pointer to it.

NOTES: This method is needed to provide the CMap (see page 186) class a uniform interface between containers. It must NEVER be used outside of the CMap (see page 186) class.

2.5.3.11.3.9 - CVList::FindSorted Method

Performs a binary search for an element on a sorted container.

C++

```
unsigned int FindSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN const _Type& rElement	The element to find.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

When found, the index of the element, the index of the first unused element otherwise.

Description

Performs a binary search for rElement on a sorted container returning the index of the first occurrence. The pfnCompare pointer can be used to change the algorithm behaviour. The behaviour is undefined if the container has not been sorted.

2.5.3.11.3.10 - GetAt**2.5.3.11.3.10.1 - CVList::GetAt Method**

Returns the element at the specified index.

C++

```
_Type& GetAt(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.11.3.10.2 - CVList::GetAt Method

Returns the element at the specified index.

C++

```
const _Type& GetAt(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.11.3.11 - CVList::GetCapacity Method

Returns the capacity.

C++

```
unsigned int GetCapacity() const;
```

Returns

The capacity.

Description

Returns the capacity, which is the number of elements already allocated that may or may not be already in use.

2.5.3.11.3.12 - CVList::GetEndIndex Method

Returns the index of the first unused element.

C++

```
unsigned int GetEndIndex() const;
```

Returns

The index of the first unused element.

Description

Returns the index of the first unused element.

2.5.3.11.3.13 - CVList::GetFirstIndex Method

Returns the index of the first used element.

C++

```
unsigned int GetFirstIndex() const;
```

Returns

The index of the first used element.

Description

Returns the index of the first used element.

2.5.3.11.3.14 - CVList::GetLastIndex Method

Returns the index of the last used element.

C++

```
unsigned int GetLastIndex() const;
```

Returns

The index of the last used element.

Description

Returns the index of the last used element.

2.5.3.11.3.15 - CVList::GetLockCapacity Method

Returns the lock capacity count.

C++

```
unsigned int GetLockCapacity() const;
```

Returns

The lock capacity count.

Description

Returns the lock capacity count. The lock capacity count is a counter that is increased each time LockCapacity (see page 256) is called and decreased each time UnlockCapacity (see page 258) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 256) or ReserveCapacity (see page 256) fails.

2.5.3.11.3.16 - CVList::GetMaxElementIndex Method

Performs a sequential search for the "biggest" element.

C++

```
unsigned int GetMaxElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "biggest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "biggest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.11.3.17 - CVList::GetMinElementIndex Method

Performs a sequential search for the "smallest" element.

C++

```
unsigned int GetMinElementIndex(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement,
IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0) const;
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

The index of the "smallest" element found in the container. An empty container always returns 0.

Description

Performs a sequential search for the "smallest" element of an unsorted container. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.11.3.18 - CVList::GetSize Method

Gets the size of the CVList (see page 244).

C++

```
unsigned int GetSize() const;
```

Returns

The size.

Description

Returns the size, which is the number of elements that are allocated and already in use.

2.5.3.11.3.19 - Insert**2.5.3.11.3.19.1 - CVList::Insert Method**

Inserts the elements contained within another CVList (see page 244).

C++

```
mxt_result Insert(IN unsigned int uIndex, IN const CVList& rVList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the elements.
IN const CVList& rVList	A VList that contains the elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts the elements contained within another CVList (see page 244) to a specified index. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.19.2 - CVList::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the first element.
IN unsigned int uCount	The number of elements to insert.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.19.3 - CVList::Insert Method

Inserts one or more elements.

C++

```
mxt_result Insert(IN unsigned int uIndex, IN unsigned int uCount, IN const _Type& rElement);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to insert the first element.
IN unsigned int uCount	The number of elements to insert.
IN const _Type& rElement	A reference to an element that is used to construct the elements.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one or more elements and constructs them from rElement. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.20 - CVList::InsertSorted Method

Inserts one element, leaving the container sorted.

C++

```
mxt_result InsertSorted(IN const _Type& rElement, IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN const _Type& rElement	A reference to an element that is used to construct the new element.
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Inserts one element and constructs it from rElement, leaving the container sorted. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new element. The lock capacity count is still enforced and if it is not zero, the insertion fails. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.11.3.21 - CVList::IsEmpty Method

Checks if the CVList (see page 244) is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the size is 0.

Description

Returns true if the size is 0; in other words, if no elements are currently in use.

2.5.3.11.3.22 - CVList::IsFull Method

Checks if the CVList (see page 244) is full.

C++

```
bool IsFull() const;
```

Returns

True when the container is full according to its capacity.

Description

This method returns true when the size of the container equals its capacity, i.e. there's no more room to add new elements without allocating more memory.

2.5.3.11.3.23 - CVList::LockCapacity Method

Locks the capacity.

C++

```
void LockCapacity();
```

Description

Locks the capacity. The lock capacity count is a counter that is increased each time LockCapacity is called and decreased each time UnlockCapacity (see page 258) is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 256) or ReserveCapacity (see page 256) fails.

2.5.3.11.3.24 - CVList::Merge Method

Moves all the elements of another VList to a specific index.

C++

```
mxt_result Merge(IN unsigned int uIndex, INOUT CVList& rVList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
INOUT CVList& rVList	The source CVList (see page 244).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Moves all the elements of another CVList (see page 244) to uIndex. The capacity is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.25 - CVList::ReduceCapacity Method

Reduces the capacity.

C++

```
mxt_result ReduceCapacity(IN unsigned int uDownToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uDownToCapacity	The wanted capacity.

Returns

resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY

Description

Reduces the capacity. The capacity is reduced to uDownToCapacity if the size of the CVList (see page 244) is less than uDownToCapacity and if uDownToCapacity is less than the current capacity. This method fails if the lock capacity count is not 0.

2.5.3.11.3.26 - CVList::ReserveCapacity Method

Increases the capacity.

C++

```
mxt_result ReserveCapacity(IN unsigned int uUpToCapacity);
```

Parameters

Parameters	Description
IN unsigned int uUpToCapacity	The wanted capacity.

Returns

```
resS_OK resFE_INVALID_STATE resFE_OUT_OF_MEMORY
```

Description

Increases the capacity. The capacity is increased to uUpToCapacity if uUpToCapacity is greater than the current capacity. This method fails if the lock capacity count is not 0 and the wanted capacity is greater than the current capacity.

2.5.3.11.3.27 - CVList::SetComparisonFunction Method

Sets the default comparison function for this container.

C++

```
mxt_result SetComparisonFunction(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq), IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque comparison parameter.
pfnCompare	Pointer to a comparison function.

Returns

```
resS_OK
```

Description

Sets the default comparison function to be used in Search/Sort (see page 257) algorithms for this container instance. The comparison function needs to return one of the following values: < 0 if rOneElement is "smaller" than rOtherElement. > 0 if rOneElement is "bigger" than rOtherElement. 0 if rOneElement and rOtherElement are equivalent.

2.5.3.11.3.28 - CVList::Sort Method

Sorts the elements in the container using the shellsort algorithm.

C++

```
void Sort(IN int (pfnCompare)(IN const _Type& rOneElement, IN const _Type& rOtherElement, IN mxt_opaque opq) = NULL, IN mxt_opaque opq = 0);
```

Parameters

Parameters	Description
IN mxt_opaque opq = 0	An opaque to be passed to the pfnCompare. Ignored if pfnCompare is NULL.
pfnCompare	An optional _Type comparison function or NULL.

Description

Sorts the elements in the container using the shellsort algorithm. The pfnCompare pointer can be used to change the algorithm behaviour.

2.5.3.11.3.29 - CVList::Split Method

Moves some elements from this VList to another starting from a specific index.

C++

```
mxt_result Split(IN unsigned int uIndex, OUT CVList& rVList);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index where to move the elements.
OUT CVList& rVList	The destination CVList (see page 244).

Returns

resS_OK resFE_OUT_OF_MEMORY

Description

Moves some elements from this CVList (see page 244) to another starting from a specific index. The destination vector is emptied first. The capacity of the target vector is increased if there are not enough allocated elements to accommodate the insertion of the new elements. The lock capacity count is still enforced and if it is not zero, the insertion fails.

2.5.3.11.3.30 - CVList::Swap Method

Exchanges the position of two elements.

C++

```
void Swap(IN unsigned int uFirstIndex, IN unsigned int uSecondIndex);
```

Parameters

Parameters	Description
IN unsigned int uFirstIndex	The index of the first element to swap.
IN unsigned int uSecondIndex	The index of the second element to swap.

Description

Exchanges the position of two elements.

2.5.3.11.3.31 - CVList::UnlockCapacity Method

Unlocks the capacity.

C++

```
void UnlockCapacity();
```

Description

Unlocks the capacity. The lock capacity count is a counter that is increased each time LockCapacity (see page 256) is called and decreased each time UnlockCapacity is called. If the lock capacity count is greater than zero, this means that any call to ReduceCapacity (see page 256) or ReserveCapacity (see page 256) fails.

2.5.3.11.4 - Operators

2.5.3.11.4.1 - CVList::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CVList<_Type>& rVList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rVList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if both CVList (see page 244) are different, otherwise returns false.

Description

Verifies that the left hand CVList (see page 244) is different than the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11.4.2 - []

2.5.3.11.4.2.1 - CVList::[] Operator

Returns the element at the specified index.

C++

```
_Type& operator [](IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.11.4.2.2 - CVList::[] Operator

Returns the element at the specified index.

C++

```
const _Type& operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	Index of the element to return.

Returns

The element at uIndex.

Description

Returns the element at uIndex. Returns an invalid element if uIndex is equal to or greater than the size.

2.5.3.11.4.3 - CVList::< Operator

Less than operator.

C++

```
bool operator <(IN const CVList<_Type>& rvlstList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if the left hand CVList (see page 244) is smaller than the right hand CVList (see page 244), otherwise returns false.

Description

Verifies if the left hand CVList (see page 244) is less than the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11.4.4 - CVList::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CVList<_Type>& rvlstList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if the left hand CVList (see page 244) is smaller than or equal to the right hand CVList (see page 244), otherwise returns false

Description

Verifies if the left hand CVList (see page 244) is less than or equal to the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11.4.5 - CVList::= Operator

Assignment operator.

C++

```
CVList<_Type>& operator =(IN const CVList<_Type>& rvlstList);
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to assign.

Returns

The assigned CVlist.

Description

Assigns the right hand CVList (see page 244) to the left hand CVList (see page 244).

2.5.3.11.4.6 - CVList::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CVList<_Type>& rvlstList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if both CVList (see page 244) are identical, otherwise returns false.

Description

Verifies that the left hand CVList (see page 244) is identical to the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11.4.7 - CVList::> Operator

Greater than operator.

C++

```
bool operator >(IN const CVList<_Type>& rvlstList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if the left hand CVList (see page 244) is greater than the right hand CVList (see page 244), otherwise returns false.

Description

Verifies if the left hand CVList (see page 244) is greater than the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.5.3.11.4.8 - CVList::>= Operator

Greater than or equal to operator.

C++

```
bool operator >=(IN const CVList<_Type>& rvlstList) const;
```

Parameters

Parameters	Description
IN const CVList<_Type>& rvlstList	Reference to the CVList (see page 244) to compare.

Returns

Returns true if the left hand CVList (see page 244) is greater than or equal to the right hand CVList (see page 244), otherwise returns false.

Description

Verifies if the left hand CVList (see page 244) is less than or equal to the right hand CVList (see page 244). This is done by comparing containers element by element and returns once an element is different from the other. This behaviour is identical to the one used in strcmp.

2.6 - Config

This section documents the Sources/Config folder of the M5T Framework. It is divided in functional subsections:

- Macros (see page 261)
- Variables (see page 318)

2.6.1 - Macros

This section documents the macros of the Sources/Config folder.

Macros

Macro	Description
MX_TRACE_NEW_BUFFER (see page 313)	Sets the default configuration for the MxTrace internal buffer.
MX_TRACE_RELEASE_BUFFER (see page 313)	Sets the default configuration for the MxTrace internal buffer.
MXD_64BITS_CUSTOM_TYPE (see page 268)	Enables a minimal 64 bits variable support.
MXD_64BITS_SUPPORT_DISABLE (see page 268)	Disables 64 bits types and methods.
MXD_ALIGNED_ACCESS_REQUIRED (see page 269)	Enables unaligned data types.
MXD_ARCH_AMD64 (see page 269)	Indicates that build is done for the AMD64 architecture.
MXD_ARCH_ARM (see page 269)	Indicates that build is done for the ARM architecture.
MXD_ARCH_IX86 (see page 269)	Indicates that build is done for the ix86 architecture.
MXD_ARCH_MIPS (see page 270)	Indicates that build is done for the MIPS architecture.
MXD_ARCH_NIOS2 (see page 270)	Indicates that build is done for the NIOS2 architecture.
MXD_ARCH_POWERPC (see page 270)	Indicates that build is done for the PowerPC architecture.
MXD_ASSERT_CALL_STACK_TRACE_DISABLE (see page 270)	Disables the display of the call stack trace.
MXD_ASSERT_CALL_STACK_TRACE_OVERRIDE (see page 270)	Indicates that the default "Call-Stack Trace Handler" is overridden at compile time.
MXD_ASSERT_DISABLE_DEBUG_BREAK (see page 271)	Disables the debug break behaviour.
MXD_ASSERT_DISABLE_OUTPUT_FILENAME (see page 271)	Preprocessor macro to disable the filename display in the trace output.
MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER (see page 271)	Preprocessor macro to disable the line number display in the trace output.
MXD_ASSERT_ENABLE_ALL (see page 272)	Assertion categories configuration macros.
MXD_ASSERT_ENABLE_RT (see page 272)	Assertion categories configuration macros.
MXD_ASSERT_ENABLE_STD (see page 272)	Assertion categories configuration macros.

<code>MXD_ASSERT_ENABLE_SUPPORT</code> (see page 272)	Assertion categories configuration macros.
<code>MXD_ASSERT_FAIL_OVERRIDE</code> (see page 272)	Indicates that the default "Assertion Failed Handler" is overridden at compile time.
<code>MXD_ASSERT_FINAL_BEHAVIOR_DISABLE</code> (see page 273)	Disables the final behaviour on assertion failure.
<code>MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL</code> (see page 273)	Enables the fatal final behaviour on assertion failure.
<code>MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE</code> (see page 273)	Indicates that the default "Final Behavior Handler" is overridden at compile time.
<code>MXD_ASSERT_TRACE_DISABLE</code> (see page 273)	Disables the tracing of the message on assertion failure.
<code>MXD_ASSERT_TRACE_OVERRIDE</code> (see page 274)	Indicates that the "Trace Handler" is overridden at compile time.
<code>MXD_astMEMORY_ALLOCATOR_POOL_INFO</code> (see page 274)	Defines the configuration for the memory pooling feature of the CMemoryAllocator (see page 480).
<code>MXD_astPOSIX_THREAD_SCHEDULING_INFO</code> (see page 275)	Sets the scheduling policy and priority that is used by CThread (see page 491).
<code>MXD_astVWORKS_THREAD_SCHEDULING_INFO</code> (see page 275)	Sets the scheduling policy and priority that is used by CThread (see page 491).
<code>MXD_astWINDOWS_THREAD_SCHEDULING_INFO</code> (see page 275)	Sets the scheduling policy and priority that is used by CThread (see page 491).
<code>MXD_ATOMIC_NATIVE_ENABLE_SUPPORT</code> (see page 276)	Enables the native support for atomic operations. By default, native atomic operations support is disabled.
<code>MXD_BIG_ENDIAN</code> (see page 276)	Indicates that build is done for a big endian architecture.
<code>MXD_CAA TREE_ENABLE_DEBUG</code> (see page 277)	Implements the debugging method PrintTree.
<code>MXD_CAP_ENABLE_SUPPORT</code> (see page 277)	Enables the support for Cap containers.
<code>MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES</code> (see page 277)	Defines the default size of the sub-allocator's memory blocks.
<code>MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT</code> (see page 277)	Enables the Cap sub-allocator feature.
<code>MXD_CAP_SUBALLOCATOR_STATISTICS_ENABLE_SUPPORT</code> (see page 278)	Enables the Cap sub-allocator statistics feature.
<code>MXD_CFILE_TRACES_ENABLE_SUPPORT</code> (see page 278)	Enables the support for CFile (see page 472) traces.
<code>MXD_CMARSHALER_ENABLE_DEBUG</code> (see page 278)	Adds debugging information into the CMarshaler (see page 117) class.
<code>MXD_COMPILER_DIAB</code> (see page 278)	Indicates that build is done by using the DIAB compiler.
<code>MXD_COMPILER_GNU_GCC</code> (see page 279)	Indicates that build is done by using the gcc compiler.
<code>MXD_COMPILER_MS_VC</code> (see page 279)	Indicates that build is done by using the MSVC compiler.
<code>MXD_COMPILER_TI_CL6X</code> (see page 279)	Indicates that build is done by using the TI CL6X compiler.
<code>MXD_CRYPTO_AES_CLASSNAME</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELOCATION</code> (see page 279)	Allows relocating the table of constants used by the AES algorithm.
<code>MXD_CRYPTO_AES_CORE_UNROLL</code> (see page 280)	Unrolls loops in aes_core.c.
<code>MXD_CRYPTO_AES_CTR_MODE_ONLY</code> (see page 280)	Disables all AES supported modes except for CTR.
<code>MXD_CRYPTO_AES_INCLUDE</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_MITOSFW</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_MOCANA_SS</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_NONE</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_OPENSSL</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_AES_OVERRIDE</code> (see page 280)	Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_ALL_MITOSFW</code> (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
<code>MXD_CRYPTO_ALL_MOCANA_SS</code> (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
<code>MXD_CRYPTO_ALL_NONE</code> (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
<code>MXD_CRYPTO_ALL_OPENSSL</code> (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
<code>MXD_CRYPTO_ALL_OVERRIDE</code> (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
<code>MXD_CRYPTO_BASE64_MITOSFW</code> (see page 282)	Enables/disables the compilation of the Base64 algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_BASE64_NONE</code> (see page 282)	Enables/disables the compilation of the Base64 algorithm in the Crypto folder for the given engine.
<code>MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME</code> (see page 282)	Enables/disables the compilation of the Diffie-Hellman algorithm in the Crypto folder for the given engine.

MXD_CRYPTO_RSA_MITOSFW (see page 285)	Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_RSA_MOCANA_SS (see page 285)	Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_RSA_NONE (see page 285)	Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_RSA_OPENSSL (see page 285)	Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_RSA_OVERRIDE (see page 285)	Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_CLASSNAME (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_INCLUDE (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_MITOSFW (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_MOCANA_SS (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_NONE (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECUREPRNG_OVERRIDE (see page 285)	Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_CLASSNAME (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_INCLUDE (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_MITOSFW (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_MOCANA_SS (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_NONE (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SECURESEED_OVERRIDE (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_CLASSNAME (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_DISABLE_LONG_MESSAGES (see page 286)	Disables the support for messages longer than 512 MB in the M5T Framework implementation of SHA-1.
MXD_CRYPTO_SHA1_INCLUDE (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_MITOSFW (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_MOCANA_SS (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_NONE (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_OPENSSL (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1_OVERRIDE (see page 287)	Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_CLASSNAME (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_INCLUDE (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_MITOSFW (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_NONE (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_OPENSSL (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA1MAC_OVERRIDE (see page 287)	Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2_CLASSNAME (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2_INCLUDE (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2_MITOSFW (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2_NONE (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.

MXD_CRYPTO_SHA2_OPENSSL (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2_OVERRIDE (see page 288)	Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_CLASSNAME (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_INCLUDE (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_MITOSFW (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_NONE (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_OPENSSL (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_SHA2MAC_OVERRIDE (see page 288)	Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_UUIDGENERATOR_CLASSNAME (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_UUIDGENERATOR_INCLUDE (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_UUIDGENERATOR_MITOSFW (see page 289)	Enables/disables the compilation of the UuidGenerator algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_UUIDGENERATOR_NONE (see page 289)	Enables/disables the compilation of the UuidGenerator algorithm in the Crypto folder for the given engine.
MXD_CRYPTO_UUIDGENERATOR_OVERRIDE (see page 286)	Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.
MXD_CSTRING_MINIMUM_BUFFER (see page 290)	Enable the CString (see page 126) minimum static buffer.
MXD_DATA_MODEL_ILP32 (see page 290)	Indicates that build is done for a ILP32 data model.
MXD_DATA_MODEL_LP64 (see page 290)	Indicates that build is done for a LP64 data model.
MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET (see page 291)	Configures the default value of the buffer offset when stack usage information is enabled.
MXD_DEFAULT_THREAD_STACK_SIZE (see page 291)	Configures the default stack size for a new thread created with CThread (see page 491).
MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE (see page 291)	Disables override of assert, assert_perror, and ASSERT.
MXD_ECOM_ENABLE_SUPPORT (see page 292)	Enables the support for the ECOM (see page 412) feature.
MXD_ENUM_ENABLE_SUPPORT (see page 292)	Enables support for ENUM.
MXD_ENUM_NAPTR_MAX_NON_TERMINAL (see page 292)	Configures the maximal number of non-terminal NAPTRs in an single ENUM request.
MXD_EVENT_NOTIFIER_ENABLE_SUPPORT (see page 292)	Enables the support for the Event Notifier feature.
MXD_FAULT_HANDLER_ENABLE_SUPPORT (see page 292)	Enables the use of the generic signal handler.
MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS (see page 293)	Controls the maximum number of memory block information copies that the CFrameworkInitializer::SFrameworkFinalizeInfo (see page 792) structure holds.
MXD_INCLUDE_NEW (see page 293)	Determines which file contains the definition of new and delete operators.
MXD_IPV6_ENABLE_SUPPORT (see page 294)	Enables IPv6 support.
MXD_IPV6_SCOPE_ID_MANDATORY_IN_ALL_ADDRESSES (see page 294)	Always specifies IPv6 scope.
MXD_KERBEROS_ENABLE_SUPPORT (see page 294)	Enables the support for Kerberos.
MXD_LIB_GNU_LIBC (see page 295)	Indicates that build is done by using the GNU LIBC library.
MXD_LIB_GNU_UCLIBC (see page 295)	Indicates that build is done by using the GNU UCLIBC library.
MXD_LITTLE_ENDIAN (see page 295)	Indicates that build is done for a little endian architecture.
MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT (see page 295)	Enables bound checking on memory allocation.
MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296)	Enables support for the memory allocator.
MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT (see page 296)	Enables the storage of additional information about each allocated memory block.
MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT (see page 296)	When defined, this macro enables the memory pooling support of the CMemoryAllocator (see page 480) class.
MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT (see page 297)	Enables memory tracking.
MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT (see page 297)	Enables concurrency protection for the memory allocator.
MXD_MEMORY_ALLOCATOR_STATISTICS_ENABLE_SUPPORT (see page 297)	Enables concurrency protection for the memory allocator.
MXD_MINIMAL_ALIGNMENT_IN_BYTES (see page 298)	Defines the minimal alignment in bytes that must be used by the Framework.
MXD_NETWORK_ENABLE_SUPPORT (see page 298)	Enables the compilation of the Network support found in the Network folder. Network is disabled by default.
MXD_OPENSSL_FIPS_140_2_ENABLE_SUPPORT (see page 281)	Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.
MXD_OS_LINUX (see page 298)	Indicates that build is done for the Linux operating system.

MXD_OS_NUCLEUS (see page 298)	Indicates that build is done for the Nucleus operating system.
MXD_OS_VXWORKS (see page 298)	Indicates that build is done for the VxWorks operating system.
MXD_OS_WINDOWS (see page 299)	Indicates that build is done for the Windows operating system.
MXD_OS_WINDOWS_CE (see page 299)	Indicates that build is done for the Windows CE operating system.
MXD_OS_WINDOWS_ENABLE_GQOS_QOS (see page 300)	Enables the compilation of the General Quality of Service support. GQoS is disabled by default. This is valid only when MXD_OS_WINDOWS (see page 299) is defined.
MXD_OS_WINDOWS_ENABLE_TC_QOS (see page 300)	Enables the compilation of the Traffic Control Quality of Service support. TC is disabled by default. This is valid only when MXD_OS_WINDOWS (see page 299) is defined.
MXD_PKG_ID_OVERRIDE (see page 300)	Enables package IDs override.
MXD_PKI_CALTERNATENAME_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CALTERNATENAME_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CAUTHORITYKEYIDENTIFIER_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CBASICCONSTRAINTS_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CBASICCONSTRAINTS_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATE_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATE_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATECHAIN_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATECHAINVALIDATION_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATEEXTENSION_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATEISSUER_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATEISSUER_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CCERTIFICATESUBJECT_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CERTIFICATECHAIN_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CEXTENDEDKEYUSAGE_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CKEYUSAGE_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CKEYUSAGE_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_CSUBJECTKEYIDENTIFIER_INCLUDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_MOCANA_SS (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_NONE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_OPENSSL (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PKI_OVERRIDE (see page 301)	Enables/disables the compilation of PKI (see page 663) for the given engine.
MXD_PORTABLE_RESOLVER_ENABLE_SUPPORT (see page 302)	Enables the portable DNS resolver.
MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS (see page 303)	Defines the portable DNS resolver maximum number of retransmissions.

MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS (see page 303)	Defines the portable DNS resolver timeout prior to retransmitting a query.
MXD_POST_CONFIG (see page 303)	Enables the use of the PostMxConfig header file.
MXD_POST_FRAMEWORKCFG (see page 303)	Enables the inclusion of "PostFrameworkCfg.h".
MXD_REGEXP_ENABLE_SUPPORT (see page 304)	Enables support for regular expressions.
MXD_RESOLVER_CACHE_CAPACITY (see page 304)	Sets the maximum number of concurrent cached DNS answers.
MXD_RESOLVER_CACHE_ENABLE_SUPPORT (see page 304)	Enables the DNS resolver's caching mechanism.
MXD_RESOLVER_CACHE_NEGATIVE_MAX_TTL_S (see page 304)	Maximum TTL value (in seconds) for negative DNS answers.
MXD_RESOLVER_CACHE_POSITIVE_MAX_TTL_S (see page 305)	Maximum TTL value (in seconds) for positive DNS answers.
MXD_RESOLVER_HOST_FILE_ENABLE_SUPPORT (see page 305)	Enables the DNS resolver's host file mechanism.
MXD_RESULT_ENABLE_ALL_ERROR_MESSAGES (see page 305)	Adds the error message strings of every packages to the compiled code.
MXD_RESULT_ENABLE_ERROR_MESSAGES (see page 305)	Activates the code allowing to set the error message strings.
MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES (see page 306)	Adds the Framework error message strings to the compiled code.
MXD_RESULT_ENABLE_SHARED_ERROR_MESSAGES (see page 306)	Enables shared error messages.
MXD_SERVICING_THREAD_ENABLE_SUPPORT (see page 306)	Enables the support for the Servicing Thread feature.
MXD_SERVICING_THREAD_MAX_CONSECUTIVE_ITERATIONS (see page 306)	Defines the maximum number of consecutive messages or timer events processed in CServicingThread (see page 771) when there are socket events to handle.
MXD_SNTP_CLIENT_ENABLE_SUPPORT (see page 307)	Enables the support for the SNTP Client.
MXD_STRING_DISABLE_REFCOUNT (see page 307)	Disables the CString (see page 126) reference counting mechanism.
MXD_SYSTEM_MEMORY_SIZE (see page 307)	Sets the size of the memory under VxWorks.
MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT (see page 307)	Forces the stack size of new threads to be fixed.
MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)	Enables stack usage information.
MXD_TIME_ENABLE_SUPPORT (see page 308)	Enables the support for Time.
MXD_TLS_CASYNCTLSSERVERSOCKET_INCLUDE (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_CASYNCTLSSOCKET_INCLUDE (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_CTLSSOCKET_CLASSNAME (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_CTLSSOCKET_INCLUDE (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_MOCANA_SS (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_NONE (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TLS_OPENSSL (see page 308)	Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.
MXD_TRACE_BACKTRACE_CAPACITY (see page 309)	Maximal backtrace capacity for call-stack traces under Linux.
MXD_TRACE_BUFFER_CAPACITY (see page 309)	Defines the maximum trace size.
MXD_TRACE_BUFFER_OVERRIDE (see page 313)	Sets the default configuration for the MxTrace internal buffer.
MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE (see page 309)	Macro that permits overriding the default call stack trace handler.
MXD_TRACE_EMPTY_MEMORY_QUEUE_ON_FINALIZE (see page 310)	Empties the tracing memory queue before finalizing the framework.
MXD_TRACE_ENABLE_ALL (see page 314)	Enables traces.
MXD_TRACE_ENABLE_MACRO_CALLSTACK (see page 314)	Enables traces.
MXD_TRACE_FORMAT_HANDLER_OVERRIDE (see page 310)	Macro that permits overriding the default trace format handler.
MXD_TRACE_MAX_LEVEL (see page 314)	Enables traces.
MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS (see page 310)	Specifies the maximum number of tracing output handlers.
MXD_TRACE_MESSAGES_PER_PERIOD (see page 310)	Specifies the number of messages outputted per period.
MXD_TRACE_NODES_ENABLED_AT_REGISTRATION (see page 311)	Default state of the tracing nodes.
MXD_TRACE_OUTPUT_HANDLER_OVERRIDE (see page 311)	Macro that permits overriding the default trace output handler.
MXD_TRACE_PERIOD_MS (see page 312)	Specifies the sleeping period in ms between outputs of the internal tracing thread.
MXD_TRACE_PROGNAME (see page 312)	Defines a program name to be added to all traces.
MXD_TRACE_QUEUE_SIZE (see page 312)	Specifies the tracing queue size in bytes.
MXD_TRACE_SEPARATOR (see page 312)	Defines the separator to be used between each header field.
MXD_TRACE_THREAD_NAME (see page 313)	Specifies the internal tracing thread name.
MXD_TRACE_THREAD_PRIORITY (see page 313)	Specifies the internal tracing thread's priority.
MXD_TRACE_THREAD_STACK_SIZE (see page 313)	Specifies the internal tracing thread's stack size.
MXD_TRACE_USE_DYNAMIC_ALLOC_BUFFER (see page 313)	Sets the default configuration for the MxTrace internal buffer.

MXD_TRACE_USE_EXTERNAL_THREAD (see page 314)	Use an external thread for the memory queue tracing mechanism.
MXD_TRACE_USE_MEMORY_QUEUE_BUFFER (see page 313)	Sets the default configuration for the MxTrace internal buffer.
MXD_TRACE_USE_STACK_BUFFER (see page 313)	Sets the default configuration for the MxTrace internal buffer.
MXD_TRACE0_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE1_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE2_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE3_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE4_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE5_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE6_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE7_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE8_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACE9_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_TRACEEX_ENABLE_SUPPORT (see page 314)	Enables traces.
MXD_uCSTRING_BLOCK_LENGTH (see page 315)	Defines the minimal allocation block size for the CString (see page 126) object.
MXD_UINT_MAX (see page 315)	Indicates the maximum value of an unsigned integer.
MXD_VXWORKS_VERSION (see page 316)	Indicates that build is done for the specified VxWorks version.
MXD_XML_DEPRECATED_ENABLE_SUPPORT (see page 316)	Enables the compilation of the deprecated XML (see page 839) support found in the Xml folder. Deprecated XML (see page 839) support is disabled by default.
MXD_XML_ENABLE_SUPPORT (see page 316)	Enables the compilation of the XML (see page 839) support found in the Xml folder. XML (see page 839) is disabled by default.
MXD_XML_PARSER_EXPAT_ENABLE_SUPPORT (see page 316)	Enables the compilation of the XML (see page 839) Expat parser support found in the Xml folder. XML (see page 839) Expat parser support is disabled by default.
MXD_ENABLE_NAMESPACE (see page 317)	Enables the use of namespaces.
MXD_OS_NONE (see page 318)	Indicates that build is done for an unspecified operating system.

2.6.1.1 - MXD_64BITS_CUSTOM_TYPE Macro

Enables a minimal 64 bits variable support.

C++

```
#define MXD_64BITS_CUSTOM_TYPE
```

Description

Preprocessor macro used to enable a minimal 64 bits variable support.

Usually, this preprocessor define is suitable when the following conditions are met:

- The MXD_OS_NONE (see page 318) preprocessor define is present.
- The framework is included in its minimal way.
- Native 64 bits data type is not available.
- Minimal 64 bits support is required.

IMPORTANT: When this define is present, the standard tracing mechanism MUST be disabled because it relies on 64 bits support. In such a case, a trace handler must be provided.

Location

Define this in "PreMxConfig.h".

See Also

MXD_OS_NONE (see page 318), MXD_64BITS_SUPPORT_DISABLE (see page 268)

2.6.1.2 - MXD_64BITS_SUPPORT_DISABLE Macro

Disables 64 bits types and methods.

C++

```
#define MXD_64BITS_SUPPORT_DISABLE
```

Description

Preprocessor macro used to disable support and inclusion of code for 64 bits methods in a minimal port of the framework.

Usually, this preprocessor define is suitable when the following conditions are met:

- The MXD_OS_NONE (see page 318) preprocessor define is present.
- The framework is included in its minimal way.
- Native 64 bits data type is not available.
- 64 bits support is NOT required.

When NOT defined, the 64 bits support is normally offered natively or only minimal support is required. See MXD_64BITS_CUSTOM_TYPE (see page 268).

IMPORTANT: When this define is present, the standard tracing mechanism MUST be disabled because it relies on 64 bits support. In such a case, a trace handler must be provided.

Location

Define this in "PreMxConfig.h".

See Also

MXD_OS_NONE (see page 318), MXD_64BITS_CUSTOM_TYPE (see page 268)

2.6.1.3 - MXD_ALIGNED_ACCESS_REQUIRED Macro

Enables unaligned data types.

C++

```
#define MXD_ALIGNED_ACCESS_REQUIRED
```

Description

Preprocessor macro used to enable unaligned types to avoid unaligned access when aligned access to memory is required. Only meaningful in combination with MXD_ARCH_OTHER_XXX, ignored in other cases.

Location

Define this in "PreMxConfig.h".

2.6.1.4 - MXD_ARCH_AMD64 Macro

Indicates that build is done for the AMD64 architecture.

C++

```
#define MXD_ARCH_AMD64
```

Description

Preprocessor macro used to enable AMD64 architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.5 - MXD_ARCH_ARM Macro

Indicates that build is done for the ARM architecture.

C++

```
#define MXD_ARCH_ARM
```

Description

Preprocessor macro used to enable ARM architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.6 - MXD_ARCH_IX86 Macro

Indicates that build is done for the ix86 architecture.

C++

```
#define MXD_ARCH_IX86
```

Description

Preprocessor macro used to enable Ix86 architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.7 - MXD_ARCH_MIPS Macro

Indicates that build is done for the MIPS architecture.

C++

```
#define MXD_ARCH_MIPS
```

Description

Preprocessor macro used to enable MIPS architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.8 - MXD_ARCH_NIOS2 Macro

Indicates that build is done for the NIOS2 architecture.

C++

```
#define MXD_ARCH_NIOS2
```

Description

Preprocessor macro used to enable NIOS2 architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.9 - MXD_ARCH_PPC Macro

Indicates that build is done for the PowerPC architecture.

C++

```
#define MXD_ARCH_PPC
```

Description

Preprocessor macro used to enable PPC architecture specific code. MXD_ARCH_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.10 - MXD_ASSERT_CALL_STACK_TRACE_DISABLE Macro

Disables the display of the call stack trace.

C++

```
#define MXD_ASSERT_CALL_STACK_TRACE_DISABLE
```

Description

When defined, the call stack trace is not displayed after an assertion failure.

Location

Define this in "PreMxConfig.h".

2.6.1.11 - MXD_ASSERT_CALL_STACK_TRACE_OVERRIDE Macro

Indicates that the default "Call-Stack Trace Handler" is overridden at compile time.

C++

```
#define MXD_ASSERT_CALL_STACK_TRACE_OVERRIDE
```

Description

This macro is used to indicate that the default call-stack trace handler is overridden at compile time. When defined, the application must define its own handler in the following way, in a single .cpp file:

```
// Must previously have included MxAssert.h
//-----
#include "Basic/MxAssert.h"

SAssertCallStackTraceHandler g_stAssertCallStackTraceHandler =
{ YourCallStackTraceHandler, 0 }
```

This avoids having to call MxAssertSetNewCallStackTraceHandler (see page 19) to set the function pointer at runtime.

Location

Define this in "PreMxConfig.h".

See Also

MxAssertSetNewCallStackTraceHandler (see page 19) mxt_PFNAssertCallStackTraceHandler (see page 87).

2.6.1.12 - MXD_ASSERT_DISABLE_DEBUG_BREAK Macro

Disables the debug break behaviour.

C++

```
#define MXD_ASSERT_DISABLE_DEBUG_BREAK
```

Description

When defined, this macro prevents the suspension of the execution and prevents the debugger from being notified.

Location

Define this in "PreMxConfig.h".

See Also

MX_ASSERT_DEBUG_BREAK

2.6.1.13 - MXD_ASSERT_DISABLE_OUTPUT_FILENAME Macro

Preprocessor macro to disable the filename display in the trace output.

C++

```
#define MXD_ASSERT_DISABLE_OUTPUT_FILENAME
```

Description

When defined, the displayed message on assertion failure does not include the filename in which the assertion has been validated.

Location

Define this in "PreMxConfig.h".

See Also

MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER (see page 271).

2.6.1.14 - MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER Macro

Preprocessor macro to disable the line number display in the trace output.

C++

```
#define MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER
```

Description

When defined, the displayed message on assertion failure does not include the line number at which the assertion has been validated.

Location

Define this in "PreMxConfig.h".

See Also

MXD_ASSERT_DISABLE_OUTPUT_FILENAME (see page 271).

2.6.1.15 - Configuring Assertion Categories

Assertion categories configuration macros.

C++

```
#define MXD_ASSERT_ENABLE_STD
#define MXD_ASSERT_ENABLE_RT
#define MXD_ASSERT_ENABLE_SUPPORT
#define MXD_ASSERT_ENABLE_ALL
```

Description

It is possible to define which categories of assertions are actually compiled through the following macros. Currently, there are two categories: Standard and Real-Time.

MXD_ASSERT_ENABLE_STD enables:

- MX_ASSERT (see page 39)
- MX_ASSERT_EX (see page 40)
- MX_ASSERT_PERROR (see page 41)
- MX_ASSERT_PERROR_EX (see page 41)
- MX_ASSERT_ONLY (see page 40)

MXD_ASSERT_ENABLE_RT enables:

- MX_ASSERT_RT (see page 42)
- MX_ASSERT_RT_EX (see page 42)
- MX_ASSERT_PERROR_RT (see page 41)
- MX_ASSERT_PERROR_RT_EX (see page 42)
- MX_ASSERT_ONLY_RT (see page 40)

MXD_ASSERT_ENABLE_SUPPORT enables:

- MX_ASSERT_ONLY (see page 40)
- MX_ASSERT_ONLY_RT (see page 40)

MXD_ASSERT_ENABLE_ALL enables:

- MXD_ASSERT_ENABLE_STD
- MXD_ASSERT_ENABLE_RT

As noted above, zero or more of these macros can be defined. If no MXD_ASSERT_ENABLE_XXX macros are defined, then no assertion information is compiled.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

Assertion Mechanism Overview (see page 5)

2.6.1.16 - MXD_ASSERT_FAIL_OVERRIDE Macro

Indicates that the default "Assertion Failed Handler" is overridden at compile time.

C++

```
#define MXD_ASSERT_FAIL_OVERRIDE
```

Description

This macro is used to indicate that the default assertion failed handler is overridden at compile time. When defined, the application must define its own handler in the following way, in a single .cpp file:

```
// Must previously have included MxAssert.h
//-----
#include "Basic/MxAssert.h"

SAssertFailHandler g_stAssertFailHandler =
{ YourAssertionFailedHandler, 0 }
```

This avoids having to call MxAssertSetNewFailHandler (see page 19) to set the function pointer at runtime, and makes sure the new handler is set early at initialization time.

Location

Define this in "PreMxConfig.h".

See Also

mxt_PFNAssertFailHandler (see page 88) MxAssertSetNewFailHandler (see page 19).

2.6.1.17 - MXD_ASSERT_FINAL_BEHAVIOR_DISABLE Macro

Disables the final behaviour on assertion failure.

C++

```
#define MXD_ASSERT_FINAL_BEHAVIOR_DISABLE
```

Description

When defined, this allows the execution to continue after an assertion failure.

Note that this is useful only when using the default "Assertion Failed Handler".

Location

Define this in "PreMxConfig.h".

2.6.1.18 - MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL Macro

Enables the fatal final behaviour on assertion failure.

C++

```
#define MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL
```

Description

When defined, causes the application to exit after an assertion failure occurs.

Note that this is useful only when using the default "Assertion Failed Handler".

Location

Define this in "PreMxConfig.h".

2.6.1.19 - MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE Macro

Indicates that the default "Final Behavior Handler" is overridden at compile time.

C++

```
#define MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE
```

Description

This macro is used to indicate that the global final behaviour handler is overridden at compile time. When defined, the application must define its own handler in the following way, in a single .cpp file:

```
// Must previously have included MxAssert.h
//-----
#include "Basic/MxAssert.h"

SAssertFinalBehaviorHandler g_stAssertFinalBehaviorHandler =
{ YourFinalBehaviorHandler, 0 }
```

This avoids having to call MxAssertSetNewFinalBehaviorHandler (see page 19) to set the function pointer at runtime.

Location

Define this in "PreMxConfig.h".

See Also

MxAssertSetNewFinalBehaviorHandler (see page 19) mxt_PFNAssertFinalBehaviorHandler (see page 88).

2.6.1.20 - MXD_ASSERT_TRACE_DISABLE Macro

Disables the tracing of the message on assertion failure.

C++

```
#define MXD_ASSERT_TRACE_DISABLE
```

Description

When defined, no message is displayed upon assertion failure. The execution can be stopped anyways if debug break is enabled.

Note that this is useful only when using the default "Assertion Failed Handler".

Location

Define this in "PreMxConfig.h".

2.6.1.21 - MXD_ASSERT_TRACE_OVERRIDE Macro

Indicates that the "Trace Handler" is overridden at compile time.

C++

```
#define MXD_ASSERT_TRACE_OVERRIDE
```

Description

This macro is used to indicate that the trace handler is overridden at compile time. When defined, the application must define its own handler in the following way, in a single .cpp file:

```
// Must previously have included MxAssert.h
//-----
#include "Basic/MxAssert.h"

SAssertTraceHandler g_stAssertTraceHandler =
{ YourAssertTraceHandler, 0 }
```

This avoids having to call MxAssertSetNewTraceHandler (see page 20) to set the function pointer at runtime.

Location

Define this in "PreMxConfig.h".

See Also

MxAssertSetNewTraceHandler (see page 20) mxt_PFNAssertTraceHandler (see page 88).

2.6.1.22 - MXD_astMEMORY_ALLOCATOR_POOL_INFO Macro

Defines the configuration for the memory pooling feature of the CMemoryAllocator (see page 480).

C++

```
#define MXD_astMEMORY_ALLOCATOR_POOL_INFO { {0, 0, 0, NULL, NULL} }
```

Description

When this preprocessor macro is defined along with the MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT (see page 296) preprocessor macro, it defines the configuration for the memory pooling mechanism of the CMemoryAllocator (see page 480). This macro must take a specific format to be valid.

The following is the description of the format:

The MXD_astMEMORY_ALLOCATOR_POOL_INFO is used as an aggregate initializer to an array of CMemoryAllocator::SMemoryPoolInfo (see page 514).

Notes

The above example omits the " multi-line wrapping because it triggers warnings on some compilers.

When examining the above example, it is clear that the parameters map to the SMemoryPoolInfo structure.

The following notes are worth pointing out about the above definition:

- The g_uMEMORY_BLOCK_OVERHEAD_SIZE (see page 521) variable holds the overhead size that the CMemoryAllocator (see page 480) allocates for memory management purposes. Here it is used to pad the lower bound and upper bound of the memory pool as it is sometimes useful for the implementation of the allocation functions (last 2 parameters).
- The last pool configured does not act as a pool at all because it has zero capacity.
- The last pool configures its own allocation subroutines.

- The lower bound is exclusive (i.e., a block size of g_uMEMORY_BLOCK_OVERHEAD_SIZE (see page 521) would not be allocated in the first pool).
- The upper bound is inclusive.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296),
 MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT (see page 296)

Example

```
CMemoryAllocator::SMemoryPoolInfo s_stPoolInformations[ ] =
  MXD_astMEMORY_ALLOCATOR_POOL_INFO;
```

As such, the macro takes the format of a list of SMemoryPoolInfo aggregate initializers, as described below:

```
#define MXD_astMEMORY_ALLOCATOR_POOL_INFO
{
  { 0 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 32 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 500, NULL, NULL },
  { 32 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 64 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 200, NULL, NULL },
  { 64 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 128 + g_uMEMORY_BLOCK_OVERHEAD_SIZE, 0, MyAllocator,
    MyDeallocator },
}
```

2.6.1.23 - Thread scheduling policy and priority definitions

Sets the scheduling policy and priority that is used by CThread (see page 491).

C++

```
#define MXD_astVXWORKS_THREAD_SCHEDULING_INFO
#define MXD_astPOSIX_THREAD_SCHEDULING_INFO
#define MXD_astWINDOWS_THREAD_SCHEDULING_INFO
```

Description

These defines provide a way to control how the scheduling works in different OSs:

VxWorks: Priorities in VxWorks are defined from 0 (highest) to 255 (lowest).

The following is the default scheduling priority definition for VxWorks: #define MXD_astVXWORKS_THREAD_SCHEDULING_INFO { {255}, /*eLOWEST */ {200}, /*eLOW */ {150}, /*eNORMAL */ {100}, /*eHIGH */ {55} /*eHIGHEST */ }

Posix: The scheduling policy attribute describes how the thread is scheduled for execution relative to the other threads in the program. A thread has one of the following scheduling policies:

- SCHED_FIFO (first-in/first-out (FIFO)) The highest priority thread runs until it blocks. If there is more than one thread with the same priority and that priority is the highest among other threads, the first thread to begin running continues until it blocks. If a thread with this policy becomes ready and it has a higher priority than the currently running thread, then it pre-empts the current thread and begins running immediately.
- SCHED_RR (round-robin (RR)) The highest priority thread runs until it blocks; however, threads of equal priority, if that priority is the highest among other threads, are time sliced (time slicing is a mechanism that ensures that every thread is allowed time to execute by pre-empting running threads at fixed intervals). If a thread with this policy becomes ready and it has a higher priority than the currently running thread, then it pre-empts the current thread and begins running immediately.
- SCHED_OTHER (also known as SCHED_FG_NP) All threads are time sliced. Under this policy, all threads receive some scheduling regardless of priority. Therefore, no thread is completely denied execution time. Nevertheless, higher priority of another scheduling policy can receive more execution time. Threads with the default scheduling policy can be denied execution time by FIFO or RR threads.

The thread priority MUST be zero at all times for SCHED_OTHER. Setting any other thread priority will result in the thread being created with a thread scheduling policy equal to 0. Refer to POSIX documentation for more details.

The following is the default scheduling priority definition for Linux:

```
#define MXD_astPOSIX_THREAD_SCHEDULING_INFO
{
  {SCHED_OTHER, 0}, /*eLOWEST */
  {SCHED_OTHER, 0}, /*eLOW */
}
```

```

    { SCHED_OTHER, 0 }, /*eNORMAL */
    { SCHED_OTHER, 0 }, /*eHIGH */
    { SCHED_OTHER, 0 } /*eHIGHEST*/
}

```

The following is the default scheduling priority definition for Nucleus:

```

#define MXD_astPOSIX_THREAD_SCHEDULING_INFO
{
    { SCHED_RR, 255 }, /*eLOWEST */
    { SCHED_RR, 200 }, /*eLOW */
    { SCHED_RR, 150 }, /*eNORMAL */
    { SCHED_RR, 100 }, /*eHIGH */
    { SCHED_RR, 55 } /*eHIGHEST*/
}

```

The following is the default scheduling priority definition for Windows:

```

#define MXD_astWINDOWS_THREAD_SCHEDULING_INFO
{
    { THREAD_PRIORITY_LOWEST }, /*eLOWEST */
    { THREAD_PRIORITY_BELOW_NORMAL }, /*eLOW */
    { THREAD_PRIORITY_NORMAL }, /*eNORMAL */
    { THREAD_PRIORITY_ABOVE_NORMAL }, /*eHIGH */
    { THREAD_PRIORITY_HIGHEST } /*eHIGHEST*/
}

```

Warning

Posix threads under Linux: as SCHED_FIFO and SCHED_RR processes can pre-empt other processes forever, only root processes are allowed to activate these policies.

FOR ALL FRAMEWORK (see page 2) VERSIONS GREATER THAN 2.0.7.11, THE DEFAULT POSIX THREAD SCHEDULER UNDER THE LINUX OPERATING SYSTEM IS DEFINED TO "SCHED_OTHER". IF A USER WANTS TO HAVE THE EXACT SAME BEHAVIOUR AND PERFORMANCE AS BEFORE, THE MXD_astPOSIX_THREAD_SCHEDULING_INFO MUST BE SET TO THE "ROUND-ROBIN" THREAD POLICY AS IN THE FOLLOWING EXAMPLE:

```

#define MXD_astPOSIX_THREAD_SCHEDULING_INFO
{
    { SCHED_RR, 10 }, /*eLOWEST */
    { SCHED_RR, 30 }, /*eLOW */
    { SCHED_RR, 50 }, /*eNORMAL */
    { SCHED_RR, 70 }, /*eHIGH */
    { SCHED_RR, 90 } /*eHIGHEST*/
}

```

Location

Define this in PreFrameworkCfg.h to change the default behaviour.

2.6.1.24 - MXD_ATOMIC_NATIVE_ENABLE_SUPPORT Macro

Enables the native support for atomic operations. By default, native atomic operations support is disabled.

C++

```
#define MXD_ATOMIC_NATIVE_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined, native atomic operations are enabled in the CAtomic.h classes. If this macro is defined, and native atomic operations is not currently supported on the configured platform, a compile time error is generated. Refer to the CAtomicOperations (see page 515) class for more information on which operations are supported on which platform.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

See Also

CAtomicOperations (see page 515)

2.6.1.25 - MXD_BIG_ENDIAN Macro

Indicates that build is done for a big endian architecture.

C++

```
#define MXD_BIG_ENDIAN
```

Description

Preprocessor macro used to enable big endian architecture specific code. One of MXD_BIG_ENDIAN or MXD_LITTLE_ENDIAN (see page 295) must be defined or deduced from the compiler's information.

Location

Define this in "PreMxConfig.h".

2.6.1.26 - MXD_CAA TREE_ENABLE_DEBUG Macro

Implements the debugging method PrintTree.

C++

```
#define MXD_CAA TREE_ENABLE_DEBUG
```

Description

When this flag is defined, the following debugging method is available:

```
void CAA Tree::PrintTree( IN CAA TreeNode* pNode, IN int nIndent )
```

This method prints all nodes from the "pNode" subtree. If the tree root node is used as "pNode", the entire CAA Tree (see page 158) is printed.

Location

Define this in PreFrameworkCfg.h.

2.6.1.27 - MXD_CAP_ENABLE_SUPPORT Macro

Enables the support for Cap containers.

C++

```
#define MXD_CAP_ENABLE_SUPPORT
```

Description

Enables the support for Cap containers in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.28 - MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES Macro

Defines the default size of the sub-allocator's memory blocks.

C++

```
#define MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES 2048
```

Description

This macro specifies the number of bytes that the sub-allocator must minimally allocate when it needs a new memory block. This defaults to 2048 bytes.

Location

Define this in PreFrameworkCfg.h.

2.6.1.29 - MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT Macro

Enables the Cap sub-allocator feature.

C++

```
#define MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT
```

Description

The Cap package offers a sub-allocator. This sub-allocator pre-allocates a big memory block of MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES (see page 277) or bigger and then sub-allocates memory within that memory block. When memory blocks are filled, new ones are created.

Location

Define this in PreFrameworkCfg.h.

2.6.1.30 - MXD_CAP_SUBALLOCATOR_STATISTICS_ENABLE_SUPPORT Macro

Enables the Cap sub-allocator statistics feature.

C++

```
#define MXD_CAP_SUBALLOCATOR_STATISTICS_ENABLE_SUPPORT
```

Description

The Cap package offers a sub-allocator. This sub-allocator computes a few statistics that can be printed to screen. These statistics are tailored to help optimizing the MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES (see page 277) value for the specific system in which it is used.

Location

Define this in PreFrameworkCfg.h.

2.6.1.31 - MXD_CFILE_TRACES_ENABLE_SUPPORT Macro

Enables the support for CFile (see page 472) traces.

C++

```
#define MXD_CFILE_TRACES_ENABLE_SUPPORT
```

Description

Enables the support for CFile (see page 472) traces in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.32 - MXD_CMARSHALER_ENABLE_DEBUG Macro

Adds debugging information into the CMarshaler (see page 117) class.

C++

```
#define MXD_CMARSHALER_ENABLE_DEBUG
```

Description

This flag affects the behaviour of the CMarshaler (see page 117) class.

Class CMarshaler (see page 117): when an element is marshaled using the "<<" operator, an additional constant value indicating the type of the element is inserted with the element. When the element is extracted, the constant value is also extracted and checked. This allows to validate that the right overloaded ">>" operator has been used.

Warning

Make sure the assertion mechanism is active when you use this flag.

Location

Define this in PreFrameworkCfg.h.

See Also

CMarshaler (see page 117).

2.6.1.33 - MXD_COMPILER_DIAB Macro

Indicates that build is done by using the DIAB compiler.

C++

```
#define MXD_COMPILER_DIAB
```

Description

Preprocessor macro used to enable DIAB compiler specific code. MXD_COMPILER_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.34 - MXD_COMPILER_GNU_GCC Macro

Indicates that build is done by using the gcc compiler.

C++

```
#define MXD_COMPILER_GNU_GCC
```

Description

Preprocessor macro used to enable gnu gcc compiler specific code. MXD_COMPILER_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.35 - MXD_COMPILER_MS_VC Macro

Indicates that build is done by using the MSVC compiler.

C++

```
#define MXD_COMPILER_MS_VC
```

Description

Preprocessor macro used to enable MSVC compiler specific code. MXD_COMPILER_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.36 - MXD_COMPILER_TI_CL6X Macro

Indicates that build is done by using the TI CL6X compiler.

C++

```
#define MXD_COMPILER_TI_CL6X
```

Description

Preprocessor macro used to enable cl6x (Code Composer Studio) compiler specific code. MXD_COMPILER_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.37 - MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELOCATION Macro

Allows relocating the table of constants used by the AES algorithm.

C++

```
#define MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELOCATION
```

Description

When this switch is defined, all accesses to the constant table used by the AES algorithm are done through a base pointer. This thus allows relocating the constant table at run time.

In order to ease this relocation, this switch also enables the definition of two new global variables, g_pstAesConstTable and g_stAesConstTableSize, which respectively hold the address of the constant table and its size. This configuration is valid for MXD_CRYPTO_AES_MITOSFW (see page 280) only.

Warning

Every access to the AES constants needs to dereference a pointer instead of being resolved at compile time when MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELOCATION is defined. Consequently, defining this switch decreases the performance of the AES algorithm and is not recommended until you really need it.

Location

Define this in PreFrameworkCfg.h.

2.6.1.38 - MXD_CRYPTO_AES_CORE_UNROLL Macro

Unrolls loops in aes_core.c.

C++

```
#define MXD_CRYPTO_AES_CORE_UNROLL
```

Description

This define allows to unroll loops in the file aes_core.c. MXD_CRYPTO_AES_MITOSFW (see page 280) must be defined to have this optimization turned on. This configuration is valid for MXD_CRYPTO_AES_MITOSFW (see page 280) only.

Location

Define this in PreFrameworkCfg.h.

See Also

AES crypto algorithm configuration macros (see page 280)

2.6.1.39 - MXD_CRYPTO_AES_CTR_MODE_ONLY Macro

Disables all AES supported modes except for CTR.

C++

```
#define MXD_CRYPTO_AES_CTR_MODE_ONLY
```

Description

This disables all AES supported modes except for CTR. This configuration is valid for MXD_CRYPTO_AES_MITOSFW (see page 280) only.

Location

Define this in PreFrameworkCfg.h.

2.6.1.40 - AES crypto algorithm configuration macros

Enables/disables the compilation of the AES algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_AES_NONE
#define MXD_CRYPTO_AES_MITOSFW
#define MXD_CRYPTO_AES_OPENSSL
#define MXD_CRYPTO_AES_MOCANA_SS
#define MXD_CRYPTO_AES_OVERRIDE
#define MXD_CRYPTO_AES_INCLUDE
#define MXD_CRYPTO_AES_CLASSNAME
```

Description

These defines provide control on how the AES algorithm is compiled. MXD_CRYPTO_AES_XXX defines are mutually exclusive (except for MXD_CRYPTO_AES_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_AES_NONE: AES implementation files are not compiled.
- MXD_CRYPTO_AES_MITOSFW: The AES algorithm is provided by the Framework itself.
- MXD_CRYPTO_AES_OPENSSL: The AES algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_AES_MOCANA_SS: The AES algorithm uses the Mocana SS library.
- MXD_CRYPTO_AES_OVERRIDE: The AES algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_AES_INCLUDE: This define is used to specify the location of the .h file used when overriding the AES algorithm.
- MXD_CRYPTO_AES_CLASSNAME: This define is used to specify the name of the class used when overriding the AES algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_AES_CLASSNAME NameSpace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros ([see page 281](#))

2.6.1.41 - Crypto engine configuration macros

Enables/disables the compilation of all algorithms in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_ALL_NONE
#define MXD_CRYPTO_ALL_MITOSFW
#define MXD_CRYPTO_ALL_OPENSSL
#define MXD_OPENSSL_FIPS_140_2_ENABLE_SUPPORT
#define MXD_CRYPTO_ALL_MOCANA_SS
#define MXD_CRYPTO_ALL_OVERRIDE
```

Description

These defines provide control on how the Crypto folder is compiled. MXD_CRYPTO_ALL_XXX defines are mutually exclusive and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_ALL_NONE: All Crypto implementation files are not compiled.
- MXD_CRYPTO_ALL_MITOSFW: All Crypto algorithms are provided by the Framework itself.
- MXD_CRYPTO_ALL_OPENSSL: All Crypto algorithms use the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_OPENSSL_FIPS_ENABLE_SUPPORT: enables the OpenSSL library in FIPS mode. In order to set the OpenSSL library in FIPS mode, the string MXD_INITIALIZER_PARAM_OPENSSL_ACTIVATE_FIPS_140_2_MODE needs to be passed to the CFrameworkInitializer::Initialize ([see page 791](#)()) method.
- MXD_CRYPTO_ALL_MOCANA_SS: All Crypto algorithms use the Mocana SS library.
- MXD_CRYPTO_ALL_OVERRIDE: All Crypto algorithms use implementations provided via the overloading mechanism.

Code size considerations

If only a few algorithms are required, you can simply remove unused files from the build tree or make sure your linker strips the unused symbols from your executable.

Overriding crypto implementation

If, for any reason, one of the above crypto configurations is not suitable for your particular needs, an override mechanism is at your disposal. To use it, you need to create a .h and a .cpp files that implement your algorithm. The class of your implementation must at least offer the methods of the interface for that particular algorithm. You then have to specify the .h file of your implementation via the MXD_CRYPTO_XXX_INCLUDE macro, and either the MXD_CRYPTO_XXX_OVERRIDE or MXD_CRYPTO_ALL_OVERRIDE macro. You can replace only the algorithm(s) you really need to be different(s) and leave all others unchanged. One good reason for doing this would be to have an optimized implementation of AES for your particular product inside a DSP for example.

Notes

These defines replace the previous MXD_CRYPTO_ENGINE_XXX, which are no longer available. It is important that you now use the appropriate MXD_CRYPTO_ALL_XXX define to enable all algorithms using the same crypto engine, or any combination of MXD_CRYPTO_XALGOX_XXX to choose the crypto engine for a particular algorithm. The MXD_CRYPTO_ALL_XXX defines have the same behaviour as the corresponding MXD_CRYPTO_ENGINE_XXX had.

Location

Define this in PreFrameworkCfg.h.

See Also

AES crypto algorithm configuration macros ([see page 280](#)), Base64 crypto algorithm configuration macros ([see page 282](#)), Diffie-Hellman crypto algorithm configuration macros ([see page 282](#)), MD5 crypto algorithm configuration macros ([see page 282](#)), MD5 MAC crypto algorithm configuration macros ([see page 283](#)), CPrivateKey crypto algorithm configuration macros ([see page 284](#)), CPublicKey crypto algorithm configuration macros ([see page 284](#)), RSA crypto algorithm configuration macros ([see page 285](#)), SecurePrng crypto algorithm configuration macros ([see page 285](#)), SecureSeed crypto algorithm configuration macros ([see page 286](#)), SHA-1 crypto algorithm configuration macros ([see page 287](#)), SHA-1 MAC crypto algorithm configuration macros ([see page 287](#)), SHA-2 crypto algorithm configuration macros ([see page 288](#)), SHA-2 MAC crypto algorithm configuration macros ([see page 288](#)), Uuid Generator crypto algorithm configuration macros ([see page 289](#))

2.6.1.42 - Base64 crypto algorithm configuration macros

Enables/disables the compilation of the Base64 algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_BASE64_NONE
#define MXD_CRYPTO_BASE64_MITOSFW
```

Description

These defines provide control on how the Base64 algorithm is compiled. MXD_CRYPTO_BASE64_XXX defines are mutually exclusive and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_BASE64_NONE: The Base64 implementation files are not compiled.
- MXD_CRYPTO_BASE64_MITOSFW: The Base64 algorithm is provided by the Framework itself.

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.43 - Diffie-Hellman crypto algorithm configuration macros

Enables/disables the compilation of the Diffie-Hellman algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_DIFFIEHELLMAN_NONE
#define MXD_CRYPTO_DIFFIEHELLMAN_OPENSSL
#define MXD_CRYPTO_DIFFIEHELLMAN_MOCANA_SS
#define MXD_CRYPTO_DIFFIEHELLMAN_OVERRIDE
#define MXD_CRYPTO_DIFFIEHELLMAN_INCLUDE
#define MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME
```

Description

These defines provide control on how the Diffie-Hellman algorithm is compiled. MXD_CRYPTO_DIFFIEHELLMAN_XXX defines are mutually exclusive (except for MXD_CRYPTO_DIFFIEHELLMAN_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_DIFFIEHELLMAN_NONE: Diffie-Hellman implementation files are not compiled.
- MXD_CRYPTO_DIFFIEHELLMAN_OPENSSL: The Diffie-Hellman algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_DIFFIEHELLMAN_MOCANA_SS: The Diffie-Hellman algorithm uses the Mocana SS library.
- MXD_CRYPTO_DIFFIEHELLMAN_OVERRIDE: The Diffie-Hellman algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_DIFFIEHELLMAN_INCLUDE: This define is used to specify the location of the .h file used when overriding the Diffie-Hellman algorithm.
- MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME: This define is used to specify the name of the class used when overriding the Diffie-Hellman algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME Namespace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.44 - MD5 crypto algorithm configuration macros

Enables/disables the compilation of the MD5 algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_MD5_NONE
#define MXD_CRYPTO_MD5_MITOSFW
#define MXD_CRYPTO_MD5_OPENSSL
```

```
#define MXD_CRYPTO_MD5_MOCANA_SS
#define MXD_CRYPTO_MD5_OVERRIDE
#define MXD_CRYPTO_MD5_INCLUDE
#define MXD_CRYPTO_MD5_CLASSNAME
```

Description

These defines provide control on how the MD5 algorithm is compiled. MXD_CRYPTO_MD5_XXX defines are mutually exclusive (except for MXD_CRYPTO_MD5_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_MD5_NONE: The MD5 implementation files are not compiled.
- MXD_CRYPTO_MD5_MITOSFW: The MD5 algorithm is provided by the Framework itself.
- MXD_CRYPTO_MD5_OPENSSL: The MD5 algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_MD5_MOCANA_SS: The MD5 algorithm is undefined for use with the Mocana SS library. Use the Framework algorithm.
- MXD_CRYPTO_MD5_OVERRIDE: The MD5 algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_MD5_INCLUDE: This define is used to specify the location of the .h file used when overriding the MD5 algorithm.
- MXD_CRYPTO_MD5_CLASSNAME: This define is used to specify the name of the class used when overriding the MD5 algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_MD5_CLASSNAME Namespace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.45 - MD5 MAC crypto algorithm configuration macros

Enables/disables the compilation of the MD5 MAC algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_MD5MAC_NONE
#define MXD_CRYPTO_MD5MAC_MITOSFW
#define MXD_CRYPTO_MD5MAC_OPENSSL
#define MXD_CRYPTO_MD5MAC_MOCANA_SS
#define MXD_CRYPTO_MD5MAC_OVERRIDE
#define MXD_CRYPTO_MD5MAC_INCLUDE
#define MXD_CRYPTO_MD5MAC_CLASSNAME
```

Description

These defines provide control on how the MD5 MAC algorithm is compiled. MXD_CRYPTO_MD5MAC_XXX defines are mutually exclusive (except for MXD_CRYPTO_MD5MAC_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_MD5MAC_NONE: The MD5 MAC implementation files are not compiled.
- MXD_CRYPTO_MD5MAC_MITOSFW: The MD5 MAC algorithm is provided by the Framework itself.
- MXD_CRYPTO_MD5MAC_OPENSSL: The MD5 MAC algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_MD5MAC_MOCANA_SS: The MD5 MAC algorithm is undefined for use with the Mocana SS library. Use the Framework algorithm.
- MXD_CRYPTO_MD5MAC_OVERRIDE: The MD5 MAC algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_MD5MAC_INCLUDE: This define is used to specify the location of the .h file used when overriding the MD5 MAC algorithm.
- MXD_CRYPTO_MD5MAC_CLASSNAME: This define is used to specify the name of the class used when overriding the MD5 MAC algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_MD5MAC_CLASSNAME Namespace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.46 - CPrivateKey crypto algorithm configuration macros

Enables/disables the compilation of the CPrivateKey (see page 368) algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_PRIVATEKEY_NONE
#define MXD_CRYPTO_PRIVATEKEY_MITOSFW
#define MXD_CRYPTO_PRIVATEKEY_OPENSSL
#define MXD_CRYPTO_PRIVATEKEY_MOCANA_SS
#define MXD_CRYPTO_PRIVATEKEY_OVERRIDE
#define MXD_CRYPTO_PRIVATEKEY_INCLUDE
#define MXD_CRYPTO_PRIVATEKEY_CLASSNAME
```

Description

These defines provide control on how the CPrivateKey (see page 368) algorithm is compiled. MXD_CRYPTO_PRIVATEKEY_XXX defines are mutually exclusive (except for MXD_CRYPTO_PRIVATEKEY_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_PRIVATEKEY_NONE: The CPrivateKey (see page 368) implementation files are not compiled.
- MXD_CRYPTO_PRIVATEKEY_OPENSSL: The CPrivateKey (see page 368) algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_PRIVATEKEY_MOCANA_SS: The CPrivateKey (see page 368) algorithm uses the Mocana SS library.
- MXD_CRYPTO_PRIVATEKEY_OVERRIDE: The CPrivateKey (see page 368) algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_PRIVATEKEY_INCLUDE: This define is used to specify the location of the .h file used when overriding the CPrivateKey (see page 368) algorithm.
- MXD_CRYPTO_PRIVATEKEY_CLASSNAME: This define is used to specify the name of the class used when overriding the CPrivateKey (see page 368) algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_PRIVATEKEY_CLASSNAME NameSpace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.47 - CPublicKey crypto algorithm configuration macros

Enables/disables the compilation of the CPublicKey (see page 374) algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_PUBLICKEY_NONE
#define MXD_CRYPTO_PUBLICKEY_MITOSFW
#define MXD_CRYPTO_PUBLICKEY_OPENSSL
#define MXD_CRYPTO_PUBLICKEY_MOCANA_SS
#define MXD_CRYPTO_PUBLICKEY_OVERRIDE
#define MXD_CRYPTO_PUBLICKEY_INCLUDE
#define MXD_CRYPTO_PUBLICKEY_CLASSNAME
```

Description

These defines provide control on how the CPublicKey (see page 374) algorithm is compiled. MXD_CRYPTO_PUBLICKEY_xxx defines are mutually exclusive (except for MXD_CRYPTO_PUBLICKEY_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_PUBLICKEY_NONE: The CPublicKey (see page 374) implementation files are not compiled.
- MXD_CRYPTO_PUBLICKEY_OPENSSL: The CPublicKey (see page 374) algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_PUBLICKEY_MOCANA_SS: The CPublicKey (see page 374) algorithm uses the Mocana SS library.
- MXD_CRYPTO_PUBLICKEY_OVERRIDE: The CPublicKey (see page 374) algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_PUBLICKEY_INCLUDE: This define is used to specify the location of the .h file used when overriding the CPublicKey (see page 374) algorithm.
- MXD_CRYPTO_PUBLICKEY_CLASSNAME: This define is used to specify the name of the class used when overriding the

CPublicKey (see page 374) algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_PUBLICKEY_CLASSNAME NameSpace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.48 - RSA crypto algorithm configuration macros

Enables/disables the compilation of the RSA algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_RSA_NONE
#define MXD_CRYPTO_RSA_MITOSFW
#define MXD_CRYPTO_RSA_OPENSSL
#define MXD_CRYPTO_RSA_MOCANA_SS
#define MXD_CRYPTO_RSA_OVERRIDE
#define MXD_CRYPTO_RSA_INCLUDE
#define MXD_CRYPTO_RSA_CLASSNAME
```

Description

These defines provide control on how the RSA algorithm is compiled. MXD_CRYPTO_RSA_XXX defines are mutually exclusive (except for MXD_CRYPTO_RSA_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_RSA_NONE: The RSA implementation files are not compiled.
- MXD_CRYPTO_RSA_OPENSSL: The RSA algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_RSA_MOCANA_SS: The RSA algorithm uses the Mocana SS library.
- MXD_CRYPTO_RSA_OVERRIDE: The RSA algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_RSA_INCLUDE: This define is used to specify the location of the .h file used when overriding the RSA algorithm.
- MXD_CRYPTO_RSA_CLASSNAME: This define is used to specify the name of the class used when overriding the RSA algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_RSA_CLASSNAME NameSpace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.49 - SecurePrng crypto algorithm configuration macros

Enables/disables the compilation of the SecurePrng algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SECUREPRNG_NONE
#define MXD_CRYPTO_SECUREPRNG_MITOSFW
#define MXD_CRYPTO_SECUREPRNG_MOCANA_SS
#define MXD_CRYPTO_SECUREPRNG_OVERRIDE
#define MXD_CRYPTO_SECUREPRNG_INCLUDE
#define MXD_CRYPTO_SECUREPRNG_CLASSNAME
```

Description

These defines provide control on how the SecurePrng algorithm is compiled. MXD_CRYPTO_SECUREPRNG_XXX defines are mutually exclusive (except for MXD_CRYPTO_SECUREPRNG_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_SECUREPRNG_NONE: The SecurePrng implementation files are not compiled.
- MXD_CRYPTO_SECUREPRNG_MITOSFW: The SecurePrng algorithm is provided by the Framework itself.
- MXD_CRYPTO_SECUREPRNG_MOCANA_SS: The SecurePrng algorithm uses the Mocana SS library.
- MXD_CRYPTO_SECUREPRNG_OVERRIDE: The SecurePrng algorithm uses an implementation provided via the overloading mechanism.

- **MXD_CRYPTO_SECUREPRNG_INCLUDE**: This define is used to specify the location of the .h file used when overriding the SecurePrng algorithm.
- **MXD_CRYPTO_SECUREPRNG_CLASSNAME**: This define is used to specify the name of the class used when overriding the SecurePrng algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. `#define MXD_CRYPTO_SECUREPRNG_CLASSNAME NameSpace::ClassName`).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.50 - SecureSeed crypto algorithm configuration macros

Enables/disables the compilation of the SecureSeed algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SECURESEED_NONE
#define MXD_CRYPTO_SECURESEED_MITOSFW
#define MXD_CRYPTO_SECURESEED_MOCANA_SS
#define MXD_CRYPTO_SECURESEED_OVERRIDE
#define MXD_CRYPTO_SECURESEED_INCLUDE
#define MXD_CRYPTO_SECURESEED_CLASSNAME
#define MXD_CRYPTO_UUIDGENERATOR_OVERRIDE
#define MXD_CRYPTO_UUIDGENERATOR_INCLUDE
#define MXD_CRYPTO_UUIDGENERATOR_CLASSNAME
```

Description

These defines provide control on how the SecureSeed algorithm is compiled. MXD_CRYPTO_SECURESEED_XXX defines are mutually exclusive (except for MXD_CRYPTO_SECURESEED_INCLUDE) and at least one MUST be defined for the framework to compile.

- **MXD_CRYPTO_SECURESEED_NONE**: The SecureSeed implementation files are not compiled.
- **MXD_CRYPTO_SECURESEED_MITOSFW**: The SecureSeed algorithm is provided by the Framework itself.
- **MXD_CRYPTO_SECURESEED_MOCANA_SS**: The SecureSeed algorithm uses the Mocana SS library.
- **MXD_CRYPTO_SECURESEED_OVERRIDE**: The SecureSeed algorithm uses an implementation provided via the overloading mechanism.
- **MXD_CRYPTO_SECURESEED_INCLUDE**: This define is used to specify the location of the .h file used when overriding the SecureSeed algorithm.
- **MXD_CRYPTO_SECURESEED_CLASSNAME**: This define is used to specify the name of the class used when overriding the SecureSeed algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. `#define MXD_CRYPTO_SECURESEED_CLASSNAME NameSpace::ClassName`).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.51 - MXD_CRYPTO_SHA1_DISABLE_LONG_MESSAGES Macro

Disables the support for messages longer than 512 MB in the M5T Framework implementation of SHA-1.

C++

```
#define MXD_CRYPTO_SHA1_DISABLE_LONG_MESSAGES
```

Description

This define disables the support for messages longer than 2^{32} bits (512 MB) in the M5T Framework SHA-1 implementation. Without this define, the allowed length for the input messages is 2^{64} bits. The performance of the SHA-1 algorithm is improved on 32 bits architectures when long messages are disabled because a 32 bits integer can be used to store the message length instead of a 64 bits integer.

This define affects only the SHA-1 implementation of the M5T Framework, not the OpenSSL or custom ones.

Location

Define this in PreFrameworkCfg.h.

See Also

SHA-1 crypto algorithm configuration macros (see page 287),

2.6.1.52 - SHA-1 crypto algorithm configuration macros

Enables/disables the compilation of the SHA-1 algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SHA1_NONE
#define MXD_CRYPTO_SHA1_MITOSFW
#define MXD_CRYPTO_SHA1_OPENSSL
#define MXD_CRYPTO_SHA1_MOCANA_SS
#define MXD_CRYPTO_SHA1_OVERRIDE
#define MXD_CRYPTO_SHA1_INCLUDE
#define MXD_CRYPTO_SHA1_CLASSNAME
```

Description

These defines provide control on how the SHA-1 algorithm is compiled. MXD_CRYPTO_SHA1_XXX defines are mutually exclusive (except for MXD_CRYPTO_SHA1_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_SHA1_NONE: The SHA-1 implementation files are not compiled.
- MXD_CRYPTO_SHA1_MITOSFW: The SHA-1 algorithm is provided by the Framework itself.
- MXD_CRYPTO_SHA1_OPENSSL: The SHA-1 algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_SHA1_MOCANA_SS: The SHA-1 algorithm uses the Mocana SS library.
- MXD_CRYPTO_SHA1_OVERRIDE: The SHA-1 algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_SHA1_INCLUDE: This define is used to specify the location of the .h file used when overriding the SHA-1 algorithm.
- MXD_CRYPTO_SHA1_CLASSNAME: This define is used to specify the name of the class used when overriding the SHA-1 algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_SHA1_CLASSNAME Namespace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.53 - SHA-1 MAC crypto algorithm configuration macros

Enables/disables the compilation of the SHA-1 MAC algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SHA1MAC_NONE
#define MXD_CRYPTO_SHA1MAC_MITOSFW
#define MXD_CRYPTO_SHA1MAC_OPENSSL
#define MXD_CRYPTO_SHA1MAC_OVERRIDE
#define MXD_CRYPTO_SHA1MAC_INCLUDE
#define MXD_CRYPTO_SHA1MAC_CLASSNAME
```

Description

These defines provide control on how the SHA-1 MAC algorithm is compiled. MXD_CRYPTO_SHA1MAC_XXX defines are mutually exclusive (except for MXD_CRYPTO_SHA1MAC_INCLUDE) and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_SHA1MAC_NONE: The SHA-1 MAC implementation files are not compiled.
- MXD_CRYPTO_SHA1MAC_MITOSFW: The SHA-1 MAC algorithm is provided by the Framework itself.
- MXD_CRYPTO_SHA1MAC_OPENSSL: The SHA-1 MAC algorithm uses the OpenSSL library. Requires wcecompat.lib when used with Windows CE.
- MXD_CRYPTO_SHA1MAC_OVERRIDE: The SHA-1 MAC algorithm uses an implementation provided via the overloading mechanism.

- MXD_CRYPTO_SHA1MAC_INCLUDE: This define is used to specify the location of the .h file used when overriding the SHA-1 MAC algorithm.
- MXD_CRYPTO_SHA1MAC_CLASSNAME: This define is used to specify the name of the class used when overriding the SHA-1 MAC algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_CRYPTO_SHA1MAC_CLASSNAME Namespace::ClassName).

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.54 - SHA-2 crypto algorithm configuration macros

Enables/disables the compilation of the SHA-2 algorithm (using 256 bits only for now) in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SHA2_NONE
#define MXD_CRYPTO_SHA2_MITOSFW
#define MXD_CRYPTO_SHA2_OPENSSL
#define MXD_CRYPTO_SHA2_OVERRIDE
#define MXD_CRYPTO_SHA2_INCLUDE
#define MXD_CRYPTO_SHA2_CLASSNAME
```

Description

These defines provide control on how the SHA-2 algorithm is compiled. MXD_CRYPTO_SHA2_xxx defines are mutually exclusive (except for MXD_CRYPTO_SHA2_INCLUDE and MXD_CRYPTO_SHA2_CLASSNAME) and at least one MUST be defined in order for the Framework to compile.

- MXD_CRYPTO_SHA2_NONE: The SHA-2 implementation files are not compiled.
- MXD_CRYPTO_SHA2_MITOSFW: The SHA-2 algorithm is provided by the Framework itself.
- MXD_CRYPTO_SHA2_OPENSSL: The SHA-2 algorithm is provided by the OpenSsl library.
- MXD_CRYPTO_SHA2_OVERRIDE: The SHA-2 algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_SHA2_INCLUDE: This define is used to specify the location of the .h file used when overriding the SHA-2 algorithm.
- MXD_CRYPTO_SHA2_CLASSNAME: This define is used to specify the name of the class used when overriding the SHA-2 algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well. (i.e. #define MXD_CRYPTO_SHA2_CLASSNAME Namespace::ClassName)

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.55 - SHA-2 MAC crypto algorithm configuration macros

Enables/disables the compilation of the SHA-2 MAC algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_SHA2MAC_NONE
#define MXD_CRYPTO_SHA2MAC_MITOSFW
#define MXD_CRYPTO_SHA2MAC_OPENSSL
#define MXD_CRYPTO_SHA2MAC_OVERRIDE
#define MXD_CRYPTO_SHA2MAC_INCLUDE
#define MXD_CRYPTO_SHA2MAC_CLASSNAME
```

Description

These defines provide control on how the SHA-2 MAC algorithm is compiled. MXD_CRYPTO_SHA2MAC_XXX defines are mutually exclusive (except for MXD_CRYPTO_SHA2MAC_INCLUDE) and at least one MUST be defined in order for the Framework to compile.

- MXD_CRYPTO_SHA2MAC_NONE: The SHA-2 MAC implementation files are not compiled.
- MXD_CRYPTO_SHA2MAC_MITOSFW: The SHA-2 MAC algorithm is provided by the Framework itself.
- MXD_CRYPTO_SHA2MAC_OPENSSL: The SHA-2 MAC algorithm is provided by the OpenSSL library.
- MXD_CRYPTO_SHA2MAC_OVERRIDE: The SHA-2 MAC algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_SHA2MAC_INCLUDE: This define is used to specify the location of the .h file used when overriding the SHA-2 MAC algorithm.
- MXD_CRYPTO_SHA2MAC_CLASSNAME: This define is used to specify the name of the class used when overriding the SHA-2 MAC algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well. (i.e. #define MXD_CRYPTO_SHA2MAC_CLASSNAME NameSpace::ClassName)

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.56 - Uuid Generator crypto algorithm configuration macros

Enables/disables the compilation of the UuidGenerator algorithm in the Crypto folder for the given engine.

C++

```
#define MXD_CRYPTO_UUIDGENERATOR_NONE
#define MXD_CRYPTO_UUIDGENERATOR_MITOSFW
```

Description

These defines provide control on how the UUID generator is compiled. MXD_CRYPTO_UUIDGENERATOR_XXX defines are mutually exclusive and at least one MUST be defined for the framework to compile.

- MXD_CRYPTO_UUIDGENERATOR_NONE: The Uuid Generator implementation files are not compiled.
- MXD_CRYPTO_UUIDGENERATOR_MITOSFW: The UuidGenerator algorithm is provided by the Framework itself.
- MXD_CRYPTO_UUIDGENERATOR_OVERRIDE (see page 286): The UuidGenerator algorithm uses an implementation provided via the overloading mechanism.
- MXD_CRYPTO_UUIDGENERATOR_INCLUDE (see page 286): This define is used to specify the location of the .h file used when overriding the UuidGenerator algorithm.
- MXD_CRYPTO_UUIDGENERATOR_CLASSNAME (see page 286): This define is used to specify the name of the class used when overriding the UuidGenerator algorithm. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well. (i.e. #define MXD_CRYPTO_UUIDGENERATOR_CLASSNAME (see page 286) NameSpace::ClassName)

Location

Define this in PreFrameworkCfg.h.

See Also

Crypto engine configuration macros (see page 281)

2.6.1.57 - MXD_CSTRING_MINIMUM_BUFFER Macro

Enable the CString (see page 126) minimum static buffer.

C++

```
#define MXD_CSTRING_MINIMUM_BUFFER
```

Description

This define is used to enable the CString (see page 126) minimum static buffer. This means that all CString objects will contain a buffer inside the class of size MXD_uCSTRING_BLOCK_LENGTH (see page 315). This is useful when lots of small CString (see page 126) objects are allocated, since it bypasses dynamic memory allocation as long as the string stored in the CString (see page 126) stays smaller than MXD_uCSTRING_BLOCK_LENGTH (see page 315) characters.

Warning

MXD_STRING_DISABLE_REFCOUNT (see page 307) needs to be defined for the CString (see page 126) minimum static buffer feature to work. Should it be not defined when MXD_CSTRING_MINIMUM_BUFFER is defined, MXD_STRING_DISABLE_REFCOUNT (see page 307) will be automatically defined.

By default, the CString (see page 126) objects use dynamic memory allocation to store strings. Enabling MXD_CSTRING_MINIMUM_BUFFER will make CString (see page 126) objects that store small strings faster, but will use more memory, since a buffer of size MXD_uCSTRING_BLOCK_LENGTH (see page 315) is allocated for every CString (see page 126) object. If the string stored exceeds MXD_CSTRING_MINIMUM_BUFFER characters, dynamic memory allocation is used.

Location

Define this in PreFrameworkCfg.h.

2.6.1.58 - MXD_DATA_MODEL_ILP32 Macro

Indicates that build is done for a ILP32 data model.

C++

```
#define MXD_DATA_MODEL_ILP32
```

Description

The ILP32 data model is the default data model. It is chosen when other data models cannot be deduced from the compiler's information. Below, the data type information related to the data model:

Data type	size in bits
char	8
short	16
int	32
long	32
pointer	32

Location

Define this in "PreMxConfig.h".

2.6.1.59 - MXD_DATA_MODEL_LP64 Macro

Indicates that build is done for a LP64 data model.

C++

```
#define MXD_DATA_MODEL_LP64
```

Description

The LP64 is deduced from the compiler's information. So far, it is detected only for the following combination of compiler, OS and architecture: GCC + LINUX + AMD64. Below, the data type information related to the data model:

Data type	size in bits
char	8
short	16
int	32
long	64
pointer	64

Location

Define this in "PreMxConfig.h".

2.6.1.60 - MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET Macro

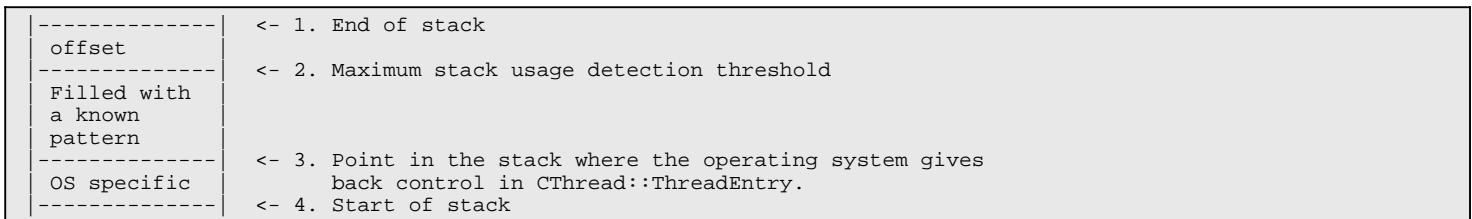
Configures the default value of the buffer offset when stack usage information is enabled.

C++

```
#define MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET 256
```

Description

When this macro is defined, it should specify the number of bytes that are subtracted from the specified stack size when filling the stack with a known pattern. Let the following be a diagram of the stack of a CThread (see page 491):



The total stack size of the thread is 1-4, but one cannot know exactly how much stack is used once the code gets in CThread::ThreadEntry because the operating system uses some of the stack for different purposes. Therefore, you need to act as if the stack was smaller to avoid buffer overflow while filling with the known pattern. The define MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET tells how much smaller the stack is.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

See Also

MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)

2.6.1.61 - MXD_DEFAULT_THREAD_STACK_SIZE Macro

Configures the default stack size for a new thread created with CThread (see page 491).

C++

```
#define MXD_DEFAULT_THREAD_STACK_SIZE
```

Description

When this macro is defined, it should specify the number of bytes that should form the stack of a new thread when it is not specified in the call to StartThread.

This parameter is used only on VxWorks and Windows and when MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT (see page 307) is defined. It has no effect in other cases

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

See Also

CThread::StartThread (see page 499)

2.6.1.62 - MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE Macro

Disables override of assert, assert_perror, and ASSERT.

C++

```
#define MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE
```

Description

By default, the assert and assert_perror macros are overridden to use the assertion mechanism. ASSERT is also undefined so calling it has no effect. The MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE macro lets the user keep the original behaviour of these three macros.

Location

Define this in "PreMxConfig.h".

2.6.1.63 - MXD_ECOM_ENABLE_SUPPORT Macro

Enables the support for the ECOM (see page 412) feature.

C++

```
#define MXD_ECOM_ENABLE_SUPPORT
```

Description

Enables the support for the ECOMs in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.64 - MXD_ENUM_ENABLE_SUPPORT Macro

Enables support for ENUM.

C++

```
#define MXD_ENUM_ENABLE_SUPPORT
```

Description

Enables support for ENUM in the Framework, as defined in RFC 3761.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.65 - MXD_ENUM_NAPTR_MAX_NON_TERMINAL Macro

Configures the maximal number of non-terminal NAPTRs in an single ENUM request.

C++

```
#define MXD_ENUM_NAPTR_MAX_NON_TERMINAL 5
```

Description

This define specifies the maximum number of non-terminal NAPTRs to lookup in a single ENUM request. Setting it to 0 effectively disables support for non-terminal NAPTRs. The default value is 5, as suggested in the draft-ietf-enum-experiences-05.txt document.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.66 - MXD_EVENT_NOTIFIER_ENABLE_SUPPORT Macro

Enables the support for the Event Notifier feature.

C++

```
#define MXD_EVENT_NOTIFIER_ENABLE_SUPPORT
```

Description

Enables the support for the Event Notifier in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.67 - MXD_FAULT_HANDLER_ENABLE_SUPPORT Macro

Enables the use of the generic signal handler.

C++

```
#define MXD_FAULT_HANDLER_ENABLE_SUPPORT
```

Description

By defining MXD_FAULT_HANDLER_ENABLE_SUPPORT, the application overrides the current behaviour of the signals defined below with an internal function. This function outputs information corresponding to the signal received and dumps the current call stack of the application. Once the call stack is dumped, the previous behaviour of the signal is restored and the signal is emitted again. For VxWorks exceptions, a hook is added to output the exception received and dumps the call stack before the exception is processed normally.

This is used for debugging purposes to output the signal information and the call stack when a signal that would normally terminate the application is received. This way, it is possible to determine what caused the signal to be emitted when it is not emitted by a user like CTRL+c or a kill -Signal PID.

Linux: This functionality is triggered by the following signals: SIGQUIT, SIGILL, SIGABRT, SIGFPE, SIGSEGV, SIGBUS, SIGSYS, and SIGTRAP. The SIGPIPE exception is blocked on Linux as it normally terminates the program. It is emitted when a program writes to a socket that has no reader.

VxWorks: This functionality is triggered by the following signals: SIGQUIT, SIGILL, SIGABRT, SIGFPE, SIGSEGV, SIGBUS, SIGTRAP, and generic exceptions.

Location

Define this in "PreFrameworkConfig.h".

See Also

PreFrameworkConfig.

2.6.1.68 - MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS Macro

Controls the maximum number of memory block information copies that the CFrameworkInitializer::SFrameworkFinalizeInfo (see page 792) structure holds.

C++

```
#define MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS
```

Description

This preprocessor macro can be defined to any number greater than 0. If left undefined, the SFrameworkFinalizeInfo structure uses a default value to initialize the memory snapshot array.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) and MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT (see page 297) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296)
MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT (see page 297)

2.6.1.69 - MXD_INCLUDE_NEW Macro

Determines which file contains the definition of new and delete operators.

C++

```
#define MXD_INCLUDE_NEW
```

Description

This macro determines which file contains the definition of the new and delete operators. In recent C++ compilers this is <new>. However, in older compilers, it may be <new.h>. Finally, some C++ compilers may automatically define these operators and therefore, no inclusions are needed.

Define this macro to:

- 0: no inclusions are made.
- 1: includes <new> (this the default behaviour, i.e., When the macro is not defined).
- 2: includes <new.h>.

Finally, if the definitions are in another file, define this macro to 0 and include the file in PreMxConfig.h (see page 317).

Location

Define this in "PreMxConfig.h".

2.6.1.70 - MXD_IPV6_ENABLE_SUPPORT Macro

Enables IPv6 support.

C++

```
#define MXD_IPV6_ENABLE_SUPPORT
```

Description

Enables support for IPv6. This is currently implemented for the following OSes:

- Linux 2.4 and up
- Windows XP SP2 and up (including 2003 server)
- Windows CE 4.1 and up

You must manually enable IPv6 support in your OS before you can use this define. For instructions on how to enable IPv6 in your OS, see the following links:

- Windows (all versions): <http://www.microsoft.com/technet/itsolutions/network/ipv6/ipv6faq.mspx>
- Linux (all versions): <http://tldp.org/HOWTO/Linux+IPv6-HOWTO/>

It is also recommended that you familiarize yourself with the following RFCs:

- RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
- RFC 2462: IPv6 Stateless Address Autoconfiguration
- RFC 3041: Extensions to IPv6 Address Autoconfiguration
- RFC 3513: IPv6 Addressing Architecture
- RFC 3596: DNS Extensions to Support IP Version 6
- RFC 3879: Deprecating Site Local Addresses
- RFC 4007: IPv6 Scoped Address Architecture
- RFC 4038: Application Aspects of IPv6 Transition
- RFC 4193: Unique Local IPv6 Unicast Addresses

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.71 - MXD_IPV6_SCOPE_ID_MANDATORY_IN_ALL_ADDRESSES Macro

Always specifies IPv6 scope.

C++

```
#define MXD_IPV6_SCOPE_ID_MANDATORY_IN_ALL_ADDRESSES
```

Description

Always specifies the scope of an IPv6 address. The scope is explicitly specified regardless of the type of the IPv6 address.

The scope is specified following RFC 4007 specification.

MXD_IPV6_ENABLE_SUPPORT (see page 294) MUST be defined to use this macro.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.72 - MXD_KERBEROS_ENABLE_SUPPORT Macro

Enables the support for Kerberos.

C++

```
#define MXD_KERBEROS_ENABLE_SUPPORT
```

Description

Enables the support for Kerberos in the Framework using the MIT Kerberos library.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

2.6.1.73 - MXD_LIB_GNU_LIBC Macro

Indicates that build is done by using the GNU LIBC library.

C++

```
#define MXD_LIB_GNU_LIBC
```

Description

Preprocessor macro used to enable GNU LIBC library specific code. MXD_LIB_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.74 - MXD_LIB_GNU_UCLIBC Macro

Indicates that build is done by using the GNU UCLIBC library.

C++

```
#define MXD_LIB_GNU_UCLIBC
```

Description

Preprocessor macro used to enable GNU UCLIBC library specific code. MXD_LIB_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.75 - MXD_LITTLE_ENDIAN Macro

Indicates that build is done for a little endian architecture.

C++

```
#define MXD_LITTLE_ENDIAN
```

Description

Preprocessor macro used to enable little endian architecture specific code. One of MXD_BIG_ENDIAN (see page 276) or MXD_LITTLE_ENDIAN must be defined or deduced from the compiler's information.

Location

Define this in "PreMxConfig.h".

2.6.1.76 - MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT Macro

Enables bound checking on memory allocation.

C++

```
#define MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined alongside MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296), a magic word is added at the beginning and at the end of each memory block allocated using MX_NEW (see page 510) or MX_NEW_ARRAY (see page 510). When a subsequent call to MX_DELETE (see page 509) or MX_DELETE_ARRAY (see page 510) is made, a verification is performed on the magic words to make sure that it has not been tampered with.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296),

2.6.1.77 - MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT Macro

Enables support for the memory allocator.

C++

```
#define MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined, calls to the MX_NEW (see page 510), MX_NEW_ARRAY (see page 510), MX_DELETE (see page 509), and MX_DELETE_ARRAY (see page 510) macros use the memory allocator implemented in the CMemoryAllocator (see page 480) class.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT (see page 295),
 MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT (see page 297),
 MXD_MEMORY_ALLOCATOR_STATISTICS_ENABLE_SUPPORT (see page 297), MX_NEW (see page 510), MX_NEW_ARRAY (see page 510), MX_DELETE (see page 509), MX_DELETE_ARRAY (see page 510).

2.6.1.78 - MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT Macro

Enables the storage of additional information about each allocated memory block.

C++

```
#define MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined alongside MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296), extra information is stored about each allocated memory block. This information contains the type of the allocated memory and the file name and line number of the allocation site as well as a bit field used to store flags.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296)
 MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT (see page 297)

2.6.1.79 - MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT Macro

When defined, this macro enables the memory pooling support of the CMemoryAllocator (see page 480) class.

C++

```
#define MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined, it enables the memory pooling mechanism of the CMemoryAllocator (see page 480). Once the memory pooling mechanism is enabled, it needs to be configured through the use of the MXD_astMEMORY_ALLOCATOR_POOL_INFO (see page 274) preprocessor macro.

Notes

If you enabled the pooling mechanism without defining a set of pools, the CMemoryAllocator (see page 480) does not use any pooling.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) MXD_astMEMORY_ALLOCATOR_POOL_INFO (see page 274)

2.6.1.80 - MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT Macro

Enables memory tracking.

C++

```
#define MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined alongside MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296), the CMemoryAllocator (see page 480) keeps track of every memory block allocated that has not been deallocated yet.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296)

MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT (see page 296)

2.6.1.81 - MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT Macro

Enables concurrency protection for the memory allocator.

C++

```
#define MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined, calls to the MX_NEW (see page 510), MX_NEW_ARRAY (see page 510), MX_DELETE (see page 509), and MX_DELETE_ARRAY (see page 510) macros are protected against concurrent access via a mutex. This is necessary when using a memory allocation library that offers no such protection.

NOTE: Defining this macro creates a dependency towards CMutex (see page 488). Make sure that it is available when enabling concurrency protection.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

2.6.1.82 - MXD_MEMORY_ALLOCATOR_STATISTICS_ENABLE_SUPPORT Macro

Enables concurrency protection for the memory allocator.

C++

```
#define MXD_MEMORY_ALLOCATOR_STATISTICS_ENABLE_SUPPORT
```

Description

When this preprocessor macro is defined, calls to the MX_NEW (see page 510), MX_NEW_ARRAY (see page 510), MX_DELETE (see page 509), and MX_DELETE_ARRAY (see page 510) macros update memory statistics, which are available via the CMemoryAllocator::GetMemoryStatistics (see page 484) method. Defining this macro also enables concurrency protection.

NOTE: Defining this macro creates a dependency towards CMutex (see page 488). Make sure that it is available when enabling memory statistics.

Notes

MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) must be enabled when using this macro.

Location

Define this in "PreMxConfig.h".

See Also

[MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT](#) (see page 296),
[MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT](#) (see page 297).

2.6.1.83 - MXD_MINIMAL_ALIGNMENT_IN_BYTES Macro

Defines the minimal alignment in bytes that must be used by the Framework.

C++

```
#define MXD_MINIMAL_ALIGNMENT_IN_BYTES 4
```

Description

This macro defines the minimal alignment in bytes that must be used by the Framework and by other application components. For example, this macro is used by the Framework memory allocator to ensure that allocated memory is always aligned adequately. Failing to do so and depending on the target architecture, unaligned memory accesses may result in notable performance degradation or even a crash. The default value is set to 4 for 32 bits platforms and to 8 for 64 bits platforms.

Location

Define this in PreFrameworkCfg.h.

2.6.1.84 - MXD_NETWORK_ENABLE_SUPPORT Macro

Enables the compilation of the Network support found in the Network folder. Network is disabled by default.

C++

```
#define MXD_NETWORK_ENABLE_SUPPORT
```

Description

This permits the compilation of the Network support found in the Network folder. This folder includes support for UDP and TCP synchronous and asynchronous sockets, as well as name resolution support. When this is not defined, the Network specific implementation files found in the Network folder are not compiled. Network is disabled by default.

Location

Define this in PreFrameworkCfg.h to change the default value.

2.6.1.85 - MXD_OS_LINUX Macro

Indicates that build is done for the Linux operating system.

C++

```
#define MXD_OS_LINUX
```

Description

Preprocessor macro used to enable Linux operating system specific code. MXD_OS_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.86 - MXD_OS_NUCLEUS Macro

Indicates that build is done for the Nucleus operating system.

C++

```
#define MXD_OS_NUCLEUS
```

Description

Preprocessor macro used to enable Nucleus operating system specific code. MXD_OS_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.87 - MXD_OS_VXWORKS Macro

Indicates that build is done for the VxWorks operating system.

C++

```
#define MXD_OS_VXWORKS
```

Description

Preprocessor macro used to enable VxWorks operating system specific code. MXD_OS_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

See Also

MXD_SYSTEM_MEMORY_SIZE (see page 307).

2.6.1.88 - MXD_OS_WINDOWS Macro

Indicates that build is done for the Windows operating system.

C++

```
#define MXD_OS_WINDOWS
```

Description

Preprocessor macro used to enable Windows operating system specific code. MXD_OS_XXX macros are mutually exclusive.

Notes

The Framework on the Windows operating system requires the Microsoft Platform Software Development Kit. The MS platform SDK contains Windows header files and Windows import libraries. The MS platform SDK can be downloaded from:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=0BAF2B35-C656-4969-ACE8-E4C0C0716ADB&displaylang=en>.

Additional documentation about the MS platform SDK can be found on: <http://msdn.microsoft.com>.

Please refer to the Microsoft documentation on how to install the Platform SDK and for any additional information.

The following MS platform SDK are required for given version of Windows OS.

- The Microsoft platform SDK 2003 (February 2003 version) is required for Microsoft Visual C++ 6.0 builds.
- The Microsoft platform SDK 2003 or later is required for Microsoft Visual Studio 7 and 8 builds.
- The Microsoft SDK v6.0 or later is required for executions under Windows Vista.

The following Include Paths must be added in Visual C++ 6/7/8 to the Tools/Options/Directories:

- Include files:
 - \${PLATFORM_SDK_INSTALL_DIR}\INCLUDE
 - \${PLATFORM_SDK_INSTALL_DIR}\INCLUDE\PRERELEASE
- Library files:
 - \${PLATFORM_SDK_INSTALL_DIR}\LIB

Replace the '\${PLATFORM_SDK_INSTALL_DIR}' part by the actual installation directory of the MS platform SDK.

Location

Define this in "PreMxConfig.h".

2.6.1.89 - MXD_OS_WINDOWS_CE Macro

Indicates that build is done for the Windows CE operating system.

C++

```
#define MXD_OS_WINDOWS_CE
```

Description

Preprocessor macro used to enable Windows CE operating system specific code. MXD_OS_XXX macros are mutually exclusive.

Location

Define this in "PreMxConfig.h".

2.6.1.90 - MXD_OS_WINDOWS_ENABLE_GQOS_QOS Macro

Enables the compilation of the General Quality of Service support. GQoS is disabled by default. This is valid only when MXD_OS_WINDOWS (see page 299) is defined.

C++

```
#define MXD_OS_WINDOWS_ENABLE_GQOS_QOS
```

Description

This allows the compilation of the GQoS support located in the Network folder. When this is not defined, the GQoS specific implementation located in the Network is not compiled. GQoS is disabled by default. The GQoS API works for layer 2 and layer 3 under Windows 2000, but only for layer 3 under Windows XP SP2. Administrative rights are not needed for it to work.

Notes

Regression have been experienced under Windows Vista, even though Microsoft states that it sill supports this API.

Location

Define this in PreFrameworkCfg.h to change the default value.

See Also

PreFrameworkCfg.h

2.6.1.91 - MXD_OS_WINDOWS_ENABLE_TC_QOS Macro

Enables the compilation of the Traffic Control Quality of Service support. TC is disabled by default. This is valid only when MXD_OS_WINDOWS (see page 299) is defined.

C++

```
#define MXD_OS_WINDOWS_ENABLE_TC_QOS
```

Description

This allows the compilation of the TC support located in the Network folder. When this is not defined, the TC specific implementation located in the Network is not compiled. TC is disabled by default. Traffic Control works for layer 2 and layer 3 tagging in both Windows 2000 and Windows XP. Administrative rights are required for it to work.

Notes

Under Windows Vista, the User Account Control needs to be disabled in order to use this API.

Location

Define this in PreFrameworkCfg.h to change the default value.

See Also

PreFrameworkCfg.h

2.6.1.92 - MXD_PKG_ID_OVERRIDE Macro

Enables package IDs override.

C++

```
#define MXD_PKG_ID_OVERRIDE
```

Description

When defined, this macro enables the application to define its own EMxPackageId (see page 12) enumerations in its PreMxConfig.h (see page 317) file. Note that this file must be found somewhere in the compiler include path.

The application must define this if it wants to use the result mechanism for its own functions/methods.

The application must make sure that the original enumeration is copied in PreMxConfig.h (see page 317) and only augment the list, otherwise it is possible that the packages will no longer compile.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

EMxPackageld (see page 12)

2.6.1.93 - Pki configuration macros

Enables/disables the compilation of PKI (see page 663) for the given engine.

C++

```
#define MXD_PKI_NONE
#define MXD_PKI_OPENSSL
#define MXD_PKI_MOCANA_SS
#define MXD_PKI_OVERRIDE
#define MXD_PKI_CCERTIFICATE_INCLUDE
#define MXD_PKI_CCERTIFICATE_CLASSNAME
#define MXD_PKI_CERTIFICATECHAIN_INCLUDE
#define MXD_PKI_CCERTIFICATECHAIN_CLASSNAME
#define MXD_PKI_CCERTIFICATECHAINVALIDATION_INCLUDE
#define MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME
#define MXD_PKI_CCERTIFICATEISSUER_INCLUDE
#define MXD_PKI_CCERTIFICATEISSUER_CLASSNAME
#define MXD_PKI_CCERTIFICATESUBJECT_INCLUDE
#define MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME
#define MXD_PKI_CCERTIFICATEEXTENSION_INCLUDE
#define MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME
#define MXD_PKI_CBASICCONSTRAINTS_INCLUDE
#define MXD_PKI_CBASICCONSTRAINTS_CLASSNAME
#define MXD_PKI_CKEYUSAGE_INCLUDE
#define MXD_PKI_CKEYUSAGE_CLASSNAME
#define MXD_PKI_CAUTHORITYKEYIDENTIFIER_INCLUDE
#define MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME
#define MXD_PKI_CALTERNATENAME_INCLUDE
#define MXD_PKI_CALTERNATENAME_CLASSNAME
#define MXD_PKI_CEXTENDEDKEYUSAGE_INCLUDE
#define MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME
#define MXD_PKI_CSUBJECTKEYIDENTIFIER_INCLUDE
#define MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME
```

Description

These defines provide control on how PKI (see page 663) is compiled. MXD_CRYPTO_AES_XXX defines are mutually exclusive (except for MXD_PKI_XXX_INCLUDE) and at least one MUST be defined for the Framework to compile.

- MXD_PKI_NONE: PKI (see page 663) implementation files are not compiled.
- MXD_PKI_OPENSSL: PKI (see page 663) uses the OpenSSL library. MXD_CRYPTO_DIFFIEHELLMAN_OPENSSL (see page 282) and MXD_CRYPTO_RSA_OPENSSL (see page 285) must also be defined when defining MXD_PKI_OPENSSL. Requires wcecompat.lib when used with Windows CE.
- MXD_PKI_MOCANA_SS: PKI (see page 663) uses the Mocana SS library. MXD_CRYPTO_DIFFIEHELLMAN_MOCANA_SS (see page 282) and MXD_CRYPTO_RSA_MOCANA_SS (see page 285) must also be defined when defining MXD_PKI_OPENSSL.
- MXD_PKI_OVERRIDE: PKI (see page 663) uses an implementation provided via the overloading mechanism.
- MXD_PKI_CCERTIFICATE_INCLUDE: This define is used to specify the location of the .h file used when overriding the CCertificate (see page 677) class.
- MXD_PKI_CCERTIFICATE_CLASSNAME: This define is used to specify the name of the class used when overriding the CCertificate (see page 677) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATE_CLASSNAME NameSpace::ClassName).
- MXD_PKI_CERTIFICATECHAIN_INCLUDE: This define is used to specify the location of the .h file used when overriding the CCertificateChain (see page 691) class.
- MXD_PKI_CCERTIFICATECHAIN_CLASSNAME: This define is used to specify the name of the class used when overriding the CCertificateChain (see page 691) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATECHAIN_CLASSNAME NameSpace::ClassName).
- MXD_PKI_CCERTIFICATECHAINVALIDATION_INCLUDE: This define is used to specify the location of the .h file used when overriding the CCertificateChainValidation (see page 696) class.
- MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME: This define is used to specify the name of the class used when overriding the CCertificateChainValidation (see page 696) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME NameSpace::ClassName).

- **MXD_PKI_CCERTIFICATEISSUER_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CCertificateIssuer (see page 704) class.
- **MXD_PKI_CCERTIFICATEISSUER_CLASSNAME**: This define is used to specify the name of the class used when overriding the CCertificateIssuer (see page 704) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATEISSUER_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CCERTIFICATESUBJECT_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CCertificateSubject (see page 707) class.
- **MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME**: This define is used to specify the name of the class used when overriding the CCertificateSubject (see page 707) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CCERTIFICATEEXTENSION_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CCertificateExtension (see page 699) class.
- **MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME**: This define is used to specify the name of the class used when overriding the CCertificateExtension (see page 699) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CBASECONSTRAINTS_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CBasicConstraints (see page 674) class.
- **MXD_PKI_CBASECONSTRAINTS_CLASSNAME**: This define is used to specify the name of the class used when overriding the CBasicConstraints (see page 674) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CBASECONSTRAINTS_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CKEYUSAGE_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CKeyUsage (see page 718) class.
- **MXD_PKI_CKEYUSAGE_CLASSNAME**: This define is used to specify the name of the class used when overriding the CKeyUsage (see page 718) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CKEYUSAGE_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CAUTHORITYKEYIDENTIFIER_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CAuthorityKeyIdentifier (see page 668) class.
- **MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME**: This define is used to specify the name of the class used when overriding the CAuthorityKeyIdentifier (see page 668) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CALTERNATENAME_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CAAlternateName (see page 664) class.
- **MXD_PKI_ALTERNATENAME_CLASSNAME**: This define is used to specify the name of the class used when overriding the CAAlternateName (see page 664) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_ALTERNATENAME_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CEXTENDEDKEYUSAGE_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CExtendedKeyUsage (see page 711) class.
- **MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME**: This define is used to specify the name of the class used when overriding the CExtendedKeyUsage (see page 711) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME NameSpace::ClassName).
- **MXD_PKI_CSUBJECTKEYIDENTIFIER_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CSubjectKeyIdentifier (see page 735) class.
- **MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME**: This define is used to specify the name of the class used when overriding the CSubjectKeyIdentifier (see page 735) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME NameSpace::ClassName).

Location

Define this in PreFrameworkCfg.h.

2.6.1.94 - MXD_PORTABLE_RESOLVER_ENABLE_SUPPORT Macro

Enables the portable DNS resolver.

C++

```
#define MXD_PORTABLE_RESOLVER_ENABLE_SUPPORT
```

Description

Enables a DNS resolver that is portable across all platforms supported by the Framework.

Location

Define this in "PrFrameworkCfg.h" or in your makefile for special builds.

2.6.1.95 - MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS Macro

Defines the portable DNS resolver maximum number of retransmissions.

C++

```
#define MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS
```

Description

Defines the number of times a nameserver will be retried prior to considering that the query has failed. The default value is 5. Each server is tried once, and then the first one is retried. The query is considered retransmitted only when a nameserver is retried (the portable resolver wrapped around the list).

Location

Define this in "PrFrameworkCfg.h" or in your makefile for special builds.

2.6.1.96 - MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS Macro

Defines the portable DNS resolver timeout prior to retransmitting a query.

C++

```
#define MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS
```

Description

Defines the portable DNS resolver timeout prior to transmitting a query to a less preferred nameserver (or to wrap around when the least preferred nameserver has timed out). The default value is 1000 ms (1 second).

Location

Define this in "PrFrameworkCfg.h" or in your makefile for special builds.

2.6.1.97 - MXD_POST_CONFIG Macro

Enables the use of the PostMxConfig header file.

C++

```
#define MXD_POST_CONFIG
```

Description

When defined, the PostMxConfig header file is included at the end of MxConfig.h.

Location

Define this in "PreMxConfig.h".

See Also

PostMxConfig.h (see page 317).

2.6.1.98 - MXD_POST_FRAMEWORKCFG Macro

Enables the inclusion of "PostFrameworkCfg.h".

C++

```
#define MXD_POST_FRAMEWORKCFG
```

Description

Enables the inclusion of "PostFrameworkCfg.h" right at the end of FrameworkCfg.h. "PostFrameworkCfg.h" is an application-provided file that can contain additional configuration options to possibly override the configuration found in PreFrameworkCfg.h and FrameworkCfg.h.

Location

Define this in PreFrameworkCfg.h or in your makefile.

See Also

PreFrameworkCfg.h

2.6.1.99 - MXD_REGEXP_ENABLE_SUPPORT Macro

Enables support for regular expressions.

C++

```
#define MXD_REGEXP_ENABLE_SUPPORT
```

Description

Enables support for regular expressions in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.100 - MXD_RESOLVER_CACHE_CAPACITY Macro

Sets the maximum number of concurrent cached DNS answers.

C++

```
#define MXD_RESOLVER_CACHE_CAPACITY
```

Description

Sets the maximum number of answers concurrently cached in memory. When this number is reached, the least recently used answer is dropped from the cache to make room for the newest answer.

An expired cached answer remains in the cache until it is accessed again, then it is removed and a query is made.

Defaults to 100.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.101 - MXD_RESOLVER_CACHE_ENABLE_SUPPORT Macro

Enables the DNS resolver's caching mechanism.

C++

```
#define MXD_RESOLVER_CACHE_ENABLE_SUPPORT
```

Description

Enables a DNS resolver's caching mechanism. This applies to all DNS queries.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.102 - MXD_RESOLVER_CACHE_NEGATIVE_MAX_TTL_S Macro

Maximum TTL value (in seconds) for negative DNS answers.

C++

```
#define MXD_RESOLVER_CACHE_NEGATIVE_MAX_TTL_S
```

Description

Sets the maximum value (in seconds) a negative DNS answer can take when it is cached.

Defaults to 5 minutes.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.103 - MXD_RESOLVER_CACHE_POSITIVE_MAX_TTL_S Macro

Maximum TTL value (in seconds) for positive DNS answers.

C++

```
#define MXD_RESOLVER_CACHE_POSITIVE_MAX_TTL_S
```

Description

Sets the maximum value (in seconds) a positive DNS answer can take when it is cached.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

Defaults to one hour.

2.6.1.104 - MXD_RESOLVER_HOST_FILE_ENABLE_SUPPORT Macro

Enables the DNS resolver's host file mechanism.

C++

```
#define MXD_RESOLVER_HOST_FILE_ENABLE_SUPPORT
```

Description

Enables the DNS resolver's host file mechanism. This applies to all DNS queries. The host file has priority over the DNS cache and the resolver.

Please refer to the CHostFile (see page 749) documentation for further information.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.105 - MXD_RESULT_ENABLE_ALL_ERROR_MESSAGES Macro

Adds the error message strings of every packages to the compiled code.

C++

```
#define MXD_RESULT_ENABLE_ALL_ERROR_MESSAGES
```

Description

When this preprocessor macro is defined, the messages associated with the results used in any packages are put in the compiled code and can then be returned by MX_RGET_MSG_STR (see page 50). If this macro is not defined, MX_RGET_MSG_STR (see page 50) might return an empty string unless a package-specific error message macro (such as MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES (see page 306)) is defined.

This macro has no effect when MXD_RESULT_ENABLE_ERROR_MESSAGES (see page 305) is not defined.

Location

Define this in "PreMxConfig.h".

See Also

MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES (see page 306), MXD_RESULT_ENABLE_ERROR_MESSAGES (see page 305).

2.6.1.106 - MXD_RESULT_ENABLE_ERROR_MESSAGES Macro

Activates the code allowing to set the error message strings.

C++

```
#define MXD_RESULT_ENABLE_ERROR_MESSAGES
```

Description

When this preprocessor macro is defined, it is possible to associate messages with the error codes by using the MX_XXX_MSG_TBL_BEGIN/END macros defined in Result.h. These messages can then be returned by MX_RGET_MSG_STR (see page 50). When this macro is not defined, MX_RGET_MSG_STR (see page 50) always returns an empty string.

Location

Define this in "PreMxConfig.h".

See Also

Result.h.

2.6.1.107 - MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES Macro

Adds the Framework error message strings to the compiled code.

C++

```
#define MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES
```

Description

When this preprocessor macro is defined, the messages associated with the results used by the Framework are put in the compiled code and can then be returned by MX_RGET_MSG_STR (see page 50). If this macro is not defined, MxResultGetMsgStr always returns an empty string for the Framework error codes. This macro has no effect when MXD_RESULT_ENABLE_ERROR_MESSAGES (see page 305) is not defined.

Location

Define this in "PreFrameworkCfg.h".

See Also

MXD_RESULT_ENABLE_ERROR_MESSAGES (see page 305).

2.6.1.108 - MXD_RESULT_ENABLE_SHARED_ERROR_MESSAGES Macro

Enables shared error messages.

C++

```
#define MXD_RESULT_ENABLE_SHARED_ERROR_MESSAGES
```

Description

When defined, the MX_RGET_MSG_STR (see page 50) method produces additional shared error messages.

2.6.1.109 - MXD_SERVICING_THREAD_ENABLE_SUPPORT Macro

Enables the support for the Servicing Thread feature.

C++

```
#define MXD_SERVICING_THREAD_ENABLE_SUPPORT
```

Description

Enables the support for the Servicing Thread in the Framework. It also controls asynchronous sockets since these are based on the Servicing Thread.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.110 - MXD_SERVICING_THREAD_MAX_CONSECUTIVE_ITERATIONS Macro

Defines the maximum number of consecutive messages or timer events processed in CServicingThread (see page 771) when there are socket events to handle.

C++

```
#define MXD_SERVICING_THREAD_MAX_CONSECUTIVE_ITERATIONS 0
```

Description

In CServicingThread::Activate the number of consecutive messages or timer events processed while there are socket events to handle is limited to the number defined through this constant if this constant is not zero.

If the constant is set to zero, all messages and timer events will be processed before processing socket events.

Location

Define this in PreFrameworkCfg.h to change the default value.

2.6.1.111 - MXD_SNTP_CLIENT_ENABLE_SUPPORT Macro

Enables the support for the SNTP Client.

C++

```
#define MXD_SNTP_CLIENT_ENABLE_SUPPORT
```

Description

Enables the support for the SNTP Client in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.112 - MXD_STRING_DISABLE_REFCOUNT Macro

Disables the CString (see page 126) reference counting mechanism.

C++

```
#define MXD_STRING_DISABLE_REFCOUNT
```

Description

This is used to disable the reference counting mechanism of CString (see page 126) objects. When this is defined, copying one CString (see page 126) into another CString (see page 126) yields CStrings each having their own buffer holding the character string.

Warning

By default, the CString (see page 126) objects implement a reference counting mechanism where two CString (see page 126) objects can share the same character buffer. Access to this buffer is not synchronized but the reference count is synchronized.

MXD_STRING_DISABLE_REFCOUNT yields an application that uses more memory.

Location

Define this in PreFrameworkCfg.h.

2.6.1.113 - MXD_SYSTEM_MEMORY_SIZE Macro

Sets the size of the memory under VxWorks.

C++

```
#define MXD_SYSTEM_MEMORY_SIZE
```

Description

Sets the size of the memory available under VxWorks. This must be specified manually since there is no reliable way to get the memory size under that OS. The value specified is a number of bytes.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

2.6.1.114 - MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT Macro

Forces the stack size of new threads to be fixed.

C++

```
#define MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT
```

Description

When this macro is defined, a fixed stack size will be specified for each new thread, and the autogrow feature will be disabled.

This parameter is used only on Linux and Solaris. It has no effects on other operating systems.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

See Also

[CThread::StartThread](#) (see page 499)

2.6.1.115 - MXD_THREAD_STACK_INFO_ENABLE_SUPPORT Macro

Enables stack usage information.

C++

```
#define MXD_THREAD_STACK_INFO_ENABLE_SUPPORT
```

Description

Enables support for detecting stack usage in [CThread](#) (see page 491) objects.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.116 - MXD_TIME_ENABLE_SUPPORT Macro

Enables the support for Time.

C++

```
#define MXD_TIME_ENABLE_SUPPORT
```

Description

Enables the support for Time in the Framework.

Location

Define this in "PreFrameworkCfg.h" or in your makefile for special builds.

2.6.1.117 - Tls configuration macros

Enables/disables the compilation of the TLS (see page 794) support located in the Tls folder.

C++

```
#define MXD_TLS_NONE
#define MXD_TLS_OPENSSL
#define MXD_TLS_MOCANA_SS
#define MXD_TLS_CTLSS_INCLUDE
#define MXD_TLS_CTLSS_CLASSNAME
#define MXD_TLS_CTLSSSESSION_INCLUDE
#define MXD_TLS_CTLSSSESSION_CLASSNAME
#define MXD_TLS_CASYNCTLSSOCKET_INCLUDE
#define MXD_TLS_CASYNCTLSSERVERSOCKET_INCLUDE
```

Description

This define provides control on how TLS (see page 794) is compiled. MXD_TLS_XXX defines are mutually exclusive (except for MXD_TLS_XXX_INCLUDE and MXD_TLS_XXX_CLASSNAME) and at least one MUST be defined for the Framework to compile.

- MXD_TLS_NONE: TLS (see page 794) support is not compiled.
- MXD_TLS_OPENSSL: TLS (see page 794) support is compiled using the OpenSSL library.
- MXD_TLS_MOCANA_SS: TLS (see page 794) support is compiled using the Mocana SS library.
- MXD_TLS_CTLSS_INCLUDE: This define is used to specify the location of the .h file used when overriding the CTls (see page 796) class.
- MXD_TLS_CTLSS_CLASSNAME: This define is used to specify the name of the class used when overriding the CTls (see page 796) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_TLS_CTLSS_CLASSNAME NameSpace::ClassName).
- MXD_TLS_CTLSSSESSION_INCLUDE: This define is used to specify the location of the .h file used when overriding the CTlsSession (see page 803) class.
- MXD_TLS_CTLSSSESSION_CLASSNAME: This define is used to specify the name of the class used when overriding the CTlsSession (see page 803) class. If that class is not inside the MXD_GNS (see page 56) namespace, specify the namespace as well (i.e. #define MXD_TLS_CTLSSSESSION_CLASSNAME NameSpace::ClassName).

- **MXD_TLS_CASYNCTLSSOCKET_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CAsyncTlsSocket class.
- **MXD_TLS_CASYNCTLSSERVERSOCKET_INCLUDE**: This define is used to specify the location of the .h file used when overriding the CAsyncTlsServerSocket

Location

Define this in PreFrameworkCfg.h.

2.6.1.118 - MXD_TRACE_BACKTRACE_CAPACITY Macro

Maximal backtrace capacity for call-stack traces under Linux.

C++

```
#define MXD_TRACE_BACKTRACE_CAPACITY 128
```

Description

When using the default call-stack trace handler, this defines the maximal backtrace capacity under Linux only. This define has no effect on other platforms.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

2.6.1.119 - MXD_TRACE_BUFFER_CAPACITY Macro

Defines the maximum trace size.

C++

```
#define MXD_TRACE_BUFFER_CAPACITY 512
```

Description

By default, with the default format handler, the traces have a limited length of 512 characters, including the trace header fields plus terminating newline and NULL characters. The size of this buffer can be modified by defining MXD_TRACE_BUFFER_CAPACITY to an integer value. Use it with care, with regards to the allocation method selected. For instance, for the default allocation method (stack allocation), a buffer of that size is allocated on the stack for each trace. For performance reasons, there is no check for buffer overload.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

mxt_PFNTraceFormatHandler (see page 89)

2.6.1.120 - MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE Macro

Macro that permits overriding the default call stack trace handler.

C++

```
#define MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE
```

Description

The MX_TRACE_CALLSTACK (see page 53) macro permits an application (and the MX_ASSERT (see page 39) module) to print the current call stack. This macro calls a default call stack trace handler that can be overridden by the application with some other handler.

Defining the macro MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE permits overriding this default call stack trace handler with one defined by the application.

When this macro is defined, the application must create a handler function that follows the mxt_PFNTraceCallStackHandler (see page 89) type and assigns it to the external variable defined in MxTrace.h:

```
#define MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE YourCallStackHandler
```

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

mxt_PFNTraceCallStackHandler (see page 89), MX_TRACE_CALLSTACK (see page 53)

2.6.1.121 - MXD_TRACE_EMPTY_MEMORY_QUEUE_ON_FINALIZE Macro

Empties the tracing memory queue before finalizing the framework.

C++

```
#define MXD_TRACE_EMPTY_MEMORY_QUEUE_ON_FINALIZE
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this allows the user to configure if the internal thread outputs all messages in the memory queue before quitting or if they are lost when the framework finalizes. This is not defined by default so the messages present in the memory queue when CFrameworkInitializer::Finalize (see page 790) is called are lost.

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.122 - MXD_TRACE_FORMAT_HANDLER_OVERRIDE Macro

Macro that permits overriding the default trace format handler.

C++

```
#define MXD_TRACE_FORMAT_HANDLER_OVERRIDE
```

Description

The framework tracing mechanism comes with a default trace format, described in mxt_PFNTraceFormatHandler (see page 89).

If this default format does not meet the application's requirements, it can be changed by defining MXD_TRACE_FORMAT_HANDLER_OVERRIDE to an application-defined trace format handler.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

General Tracing Configuration (see page 2)

2.6.1.123 - MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS Macro

Specifies the maximum number of tracing output handlers.

C++

```
#define MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS
```

Description

This macro configures the maximum number of elements that can be inserted in the output handlers buffer. The output handlers buffer is traversed when a trace needs to be output, therefore making the trace go through all registered handlers.

When no value is specified by the user, the macro is automatically set to 1.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

mxt_PFNTraceOutputHandler (see page 92)

2.6.1.124 - MXD_TRACE_MESSAGES_PER_PERIOD Macro

Specifies the number of messages outputted per period.

C++

```
#define MXD_TRACE_MESSAGES_PER_PERIOD
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this define lets the user specify the number of messages outputted per period. The default is 10.

Location

Define this in PreMxConfig.h.

See Also

MxTraceSetThreadMessagesPerPeriod (see page 508), PreMxConfig.h (see page 317)

2.6.1.125 - MXD_TRACE_NODES_ENABLED_AT_REGISTRATION Macro

Default state of the tracing nodes.

C++

```
#define MXD_TRACE_NODES_ENABLED_AT_REGISTRATION
```

Description

This allows the user to determine whether or not the traces are enabled by default during registration. A value of true means the trace nodes are enabled by default while false means they are disabled. If MXD_TRACE_NODES_ENABLED_AT_REGISTRATION is not defined, the trace nodes are enabled by default.

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.126 - MXD_TRACE_OUTPUT_HANDLER_OVERRIDE Macro

Macro that permits overriding the default trace output handler.

C++

```
#define MXD_TRACE_OUTPUT_HANDLER_OVERRIDE
```

Description

By default, the framework tracing mechanism uses std::cout to output traces. In order to allow tracing to be output to the correct destination when tracing before "main", it must be possible to override the default handler at compile time. It is however not recommended to trace before the Framework is initialized. This macro allows controlling the tracing mechanism during the initialization process, when the tracing mechanism configuration functions could not have been called. This can be accomplished by defining MXD_TRACE_OUTPUT_HANDLER_OVERRIDE to an application-defined trace output handler. Although many output handlers can be registered to output the traces at the same time, it is only possible to override with one output handler. However, more output handlers can be registered at run-time. To do this, it is necessary to have an extern declaration of the application defined output handler in the file "PostMxConfig.h (see page 317)" as well as the define MXD_TRACE_OUTPUT_HANDLER_OVERRIDE. The inclusion of the "PostMxConfig.h (see page 317)" is enabled by the macro MXD_POST_CONFIG (see page 303) in the "PreMxConfig.h (see page 317)" file.

In "PreMxConfig.h (see page 317)" add:

```
#define MXD_POST_CONFIG
```

In "PostMxConfig.h (see page 317)" add:

```
extern
void TraceOutputHandler(IN EMxTraceLevel eLevel,
                       IN EMxTraceCategory eCategory,
                       IN EMxPackageId ePackage,
                       IN const char* pszTrace,
                       IN int nMsgSize);

#define MXD_TRACE_OUTPUT_HANDLER_OVERRIDE TraceOutputHandler
```

where TraceOutputHandler is the application-defined output handler.

Location

Define this in "PostMxConfig.h" or in your makefile for special builds.

See Also

General Tracing Configuration (see page 2)

2.6.1.127 - MXD_TRACE_PERIOD_MS Macro

Specifies the sleeping period in ms between outputs of the internal tracing thread.

C++

```
#define MXD_TRACE_PERIOD_MS
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this define lets the user specify the sleeping period in ms between outputs of the internal thread. The default is 100 ms.

Location

Define this in PreMxConfig.h.

See Also

MxTraceSetThreadPeriodMs (see page 509), PreMxConfig.h (see page 317)

2.6.1.128 - MXD_TRACE_PROGNAME Macro

Defines a program name to be added to all traces.

C++

```
#define MXD_TRACE_PROGNAME ""
```

Description

An application can define this macro to a character string, which appears in one header field of the default format handler for each trace. If this is not defined, it defaults to an empty character string.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

mxt_PFNTraceFormatHandler (see page 89)

2.6.1.129 - MXD_TRACE_QUEUE_SIZE Macro

Specifies the tracing queue size in bytes.

C++

```
#define MXD_TRACE_QUEUE_SIZE
```

Description

If the tracing mechanism that uses the memory queue is chosen, this define lets the user specify the queue size in bytes. The default size is 100K.

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.130 - MXD_TRACE_SEPARATOR Macro

Defines the separator to be used between each header field.

C++

```
#define MXD_TRACE_SEPARATOR
```

Description

An application can define this macro to a character, which serves as separator between each header field of a trace. If this is not defined, it defaults to '|'.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

2.6.1.131 - MXD_TRACE_THREAD_NAME Macro

Specifies the internal tracing thread name.

C++

```
#define MXD_TRACE_THREAD_NAME
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this define lets the user specify the name of the internal thread.

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.132 - MXD_TRACE_THREAD_PRIORITY Macro

Specifies the internal tracing thread's priority.

C++

```
#define MXD_TRACE_THREAD_PRIORITY
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this define lets the user specify the priority of the internal thread. The default is the same as for CThread (see page 491).

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.133 - MXD_TRACE_THREAD_STACK_SIZE Macro

Specifies the internal tracing thread's stack size.

C++

```
#define MXD_TRACE_THREAD_STACK_SIZE
```

Description

If the tracing mechanism that uses the memory queue is chosen and the internal thread is used, this define lets the user specify the stack size of the internal thread. The default is the same as for CThread (see page 491).

Location

Define this in PreMxConfig.h.

See Also

PreMxConfig.h (see page 317)

2.6.1.134 - Tracing Internal Buffer configuration

Sets the default configuration for the MxTrace internal buffer.

C++

```
#define MXD_TRACE_USE_DYNAMIC_ALLOC_BUFFER
#define MXD_TRACE_USE_STACK_BUFFER
#define MXD_TRACE_USE_MEMORY_QUEUE_BUFFER
#define MXD_TRACE_BUFFER_OVERRIDE
#define MX_TRACE_NEW_BUFFER
#define MX_TRACE_RELEASE_BUFFER
```

Description

The tracing mechanism requires a buffer to format the trace to output. Depending on system limitations or even on implementation choices, this buffer may be required to be allocated by different means. The default behaviour was chosen to use a stack allocated buffer.

```
MXD_TRACE_USE_DYNAMIC_ALLOC_BUFFER
```

Dynamic buffer allocation mechanism. A buffer is dynamically allocated (using new and delete operators) and formatted everytime a trace needs to be outputted.

```
MXD_TRACE_USE_STACK_BUFFER
```

Static buffer allocation mechanism. A buffer is allocated on the stack and formatted everytime a trace needs to be outputted.

```
MXD_TRACE_USE_MEMORY_QUEUE_BUFFER
```

Tracing mechanism that uses a memory queue. New traces are pushed on a memory queue. If this mechanism is chosen, the user has the choice to let an internal thread de-queue the traces and call the output handler periodically. The user can decide not to use the internal thread, in which case the user is responsible of de-queuing and outputting the traces. This can be done by defining MXD_TRACE_USE_EXTERNAL_THREAD (see page 314) and calling MxTraceCallOutputHandler (see page 508).

```
MXD_TRACE_BUFFER_OVERRIDE
```

User-specified buffer allocation mechanism. This allows a user to provide his own allocation mechanism. When this is defined, the user must also define the MX_TRACE_NEW_BUFFER and MX_TRACE_RELEASE_BUFFER macros. The "x" parameter is a pointer to a char buffer.

The MX_TRACE_NEW_BUFFER and MX_TRACE_RELEASE_BUFFER will allocate and free the buffer on behalf of the trace system.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

2.6.1.135 - MXD_TRACE_USE_EXTERNAL_THREAD Macro

Use an external thread for the memory queue tracing mechanism.

C++

```
#define MXD_TRACE_USE_EXTERNAL_THREAD
```

Description

If the tracing mechanism that uses the memory queue is chosen, this allows the user to configure the mechanism to not use the provided internal thread to output the traces stored in the memory queue. If this is done, the user must call MxTraceCallOutputHandler (see page 508) to output the messages from the memory queue. This gives more flexibility to the user on how the messages from the memory queue are outputted. However, the MxTraceCallOutputHandler (see page 508) is not protected against concurrent access for performance reasons, and it is the user's responsibility that it is not called from different threads. MxTraceCallOutputHandler (see page 508) can not be called after the call to CFrameworkInitializer::Finalize (see page 790).

Location

Define this in PreMxConfig.h.

See Also

MXD_TRACE_USE_MEMORY_QUEUE_BUFFER (see page 313), MxTraceCallOutputHandler (see page 508), PreMxConfig.h (see page 317)

2.6.1.136 - Configuring Tracing Levels

Enables traces.

C++

```
#define MXD_TRACE0_ENABLE_SUPPORT
#define MXD_TRACE1_ENABLE_SUPPORT
#define MXD_TRACE2_ENABLE_SUPPORT
#define MXD_TRACE3_ENABLE_SUPPORT
#define MXD_TRACE4_ENABLE_SUPPORT
#define MXD_TRACE5_ENABLE_SUPPORT
#define MXD_TRACE6_ENABLE_SUPPORT
#define MXD_TRACE7_ENABLE_SUPPORT
#define MXD_TRACE8_ENABLE_SUPPORT
#define MXD_TRACE9_ENABLE_SUPPORT
#define MXD_TRACEX_ENABLE_SUPPORT
#define MXD_TRACE_MAX_LEVEL
#define MXD_TRACE_ENABLE_ALL
```

```
#define MXD_TRACE_ENABLE_MACRO_CALLSTACK
```

Description

It is possible to define which levels of tracing are actually compiled through the following macros.

The columns correspond to the 10 levels of tracing, as follows:
0 = eLEVEL0 | 1 = eLEVEL1 | 2 = eLEVEL2 | 3 = eLEVEL3 | 4 = eLEVEL4
5 = eLEVEL0 | 6 = eLEVEL1 | 7 = eLEVEL2 | 8 = eLEVEL3 | 9 = eLEVEL4

	0	1	2	3	4	5	6	7	8	9
++ Control each tracing macro individually										
#define MXD_TRACE0_ENABLE_SUPPORT	X									
#define MXD_TRACE1_ENABLE_SUPPORT		X								
#define MXD_TRACE2_ENABLE_SUPPORT			X							
#define MXD_TRACE3_ENABLE_SUPPORT				X						
#define MXD_TRACE4_ENABLE_SUPPORT					X					
#define MXD_TRACE5_ENABLE_SUPPORT						X				
#define MXD_TRACE6_ENABLE_SUPPORT							X			
#define MXD_TRACE7_ENABLE_SUPPORT								X		
#define MXD_TRACE8_ENABLE_SUPPORT									X	
#define MXD_TRACE9_ENABLE_SUPPORT										X
++ Control on all tracing macro level										
#define MXD_TRACE_ENABLE_ALL	X	X	X	X	X	X	X	X	X	X
++ Control on consecutive tracing macro level										
#define MXD_TRACE_MAX_LEVEL 0	X									
#define MXD_TRACE_MAX_LEVEL 1		X								
#define MXD_TRACE_MAX_LEVEL 2			X	X						
#define MXD_TRACE_MAX_LEVEL 3				X	X					
#define MXD_TRACE_MAX_LEVEL 4					X	X	X	X		
#define MXD_TRACE_MAX_LEVEL 5						X	X	X	X	
#define MXD_TRACE_MAX_LEVEL 6							X	X	X	X
#define MXD_TRACE_MAX_LEVEL 7								X	X	X
#define MXD_TRACE_MAX_LEVEL 8									X	X
#define MXD_TRACE_MAX_LEVEL 9										X
++ These do not belong to any specific level										
#define MXD_TRACE_ENABLE_MACRO_CALLSTACK	X	X	X	X	X	X	X	X	X	X
#define MXD_TRACE_ENABLE_MACRO_X	X	X	X	X	X	X	X	X	X	X

As noted above, zero or more of these macros can be defined. If no MXD_TRACE{LEVEL}_ENABLE_SUPPORT macros are defined, then no tracing information is compiled.

Location

Define this in "PreMxConfig.h" or in your makefile for special builds.

See Also

General Tracing Configuration (see page 2)

2.6.1.137 - MXD_uCSTRING_BLOCK_LENGTH Macro

Defines the minimal allocation block size for the CString (see page 126) object.

C++

```
#define MXD_uCSTRING_BLOCK_LENGTH 32
```

Description

The CString (see page 126) class holds a character buffer to keep its associated string. When this buffer is allocated or when it has to be resized to a bigger size, the CString (see page 126) allocates a number of chunks of MXD_uCSTRING_BLOCK_LENGTH bytes to fit the character string.

As an example, when compiled with the default setting of 32, a one character string uses 32 bytes while a 33 character long string takes 64 bytes, and so on.

The default value is 32.

Location

Define this in PreFrameworkCfg.h to change the default value.

2.6.1.138 - MXD_UINT_MAX Macro

Indicates the maximum value of an unsigned integer.

C++

```
#define MXD_UINT_MAX
```

Description

Preprocessor macro used to define the maximum value an unsigned integer can hold on the current architecture. MXD_UINT_MAX must be defined if it can not be deduced from the compiler's information. As an example, a 32 bits architecture has its maximum unsigned integer value set to 0xffffffff and on a 64 bits architecture, the maximum value is (2^64 - 1).

Location

Define this in "PreMxConfig.h".

2.6.1.139 - MXD_VXWORKS_VERSION Macro

Indicates that build is done for the specified VxWorks version.

C++

```
#define MXD_VXWORKS_VERSION
```

Description

Preprocessor macro used to specify the version of the VxWorks operating system for which we are building. Must take a value of 0x50400 for VxWorks 5.4 and 0x050500 for VxWorks 5.5.

Location

Define this in "PreMxConfig.h".

2.6.1.140 - MXD_XML_DEPRECATED_ENABLE_SUPPORT Macro

Enables the compilation of the deprecated XML (see page 839) support found in the Xml folder. Deprecated XML (see page 839) support is disabled by default.

C++

```
#define MXD_XML_DEPRECATED_ENABLE_SUPPORT
```

Description

This permits the compilation of the deprecated XML (see page 839) support found in the Xml folder. When this is not defined, the deprecated XML (see page 839) specific implementation files found in Xml are not compiled. Deprecated XML (see page 839) is disabled by default. While one XML (see page 839) implementation should be enabled at all times, it is possible to enable both of them at the same time.

Location

Define this in PreFrameworkCfg.h to change the default value.

See Also

PreFrameworkCfg.h

2.6.1.141 - MXD_XML_ENABLE_SUPPORT Macro

Enables the compilation of the XML (see page 839) support found in the Xml folder. XML (see page 839) is disabled by default.

C++

```
#define MXD_XML_ENABLE_SUPPORT
```

Description

This permits the compilation of the XML (see page 839) support found in the Xml folder. When this is not defined, the XML (see page 839) specific implementation files found in Xml are not compiled. XML (see page 839) is disabled by default. While one XML (see page 839) implementation should be enabled at all times, it is possible to enable both of them at the same time.

Location

Define this in PreFrameworkCfg.h to change the default value.

See Also

PreFrameworkCfg.h

2.6.1.142 - MXD_XML_PARSER_EXPAT_ENABLE_SUPPORT Macro

Enables the compilation of the XML (see page 839) Expat parser support found in the Xml folder. XML (see page 839) Expat parser support is disabled by default.

C++

```
#define MXD_XML_PARSER_EXPAT_ENABLE_SUPPORT
```

Description

This permits the compilation of the XML (see page 839) Expat parser support found in the Xml folder. When this is not defined, the XML (see page 839) Expat parser specific implementation files found in Xml are not compiled. XML (see page 839) Expat Parser is disabled by default. When disabled, it allows the user to implement the IXmlParser interface specifically for another XML (see page 839) parser and use it transparently within the XML (see page 839) specific files.

Warning

If disabled, another custom parser implementation MUST be provided and compiled for the rest of the XML (see page 839) files to work properly. This custom implementation MUST implement the IXmlParser interface, MUST use the IXmlParserMgr interface, and its ECOM (see page 412) CLSID MUST be defined as CLSID_CXmlParser. Failure to comply will most likely result in a crash.

Location

Define this in PreFrameworkCfg.h to change the default value.

See Also

PreFrameworkCfg.h

2.6.1.143 - MXD_ENABLE_NAMESPACE Macro

Enables the use of namespaces.

C++

```
#define MXD_ENABLE_NAMESPACE
```

Description

By defining MXD_ENABLE_NAMESPACE, standard namespaces are defined and namespaces are enabled on compilers that support it. Note that some debuggers might have difficulties with namespaces.

Location

Define this in "PreMxConfig.h".

See Also

MX_NAMESPACE_START (see page 47) MX_NAMESPACE_END (see page 47) MX_NAMESPACE_USE (see page 48).

2.6.1.144 - PreMxConfig.h

Configuration header file for all of M5T's software.

Description

The PreMxConfig header file is where an application configures the system-wide compilation options that affect all of the provided software. This can include configuring assertions, tracing, and other system-wide options.

Warning

Since all of M5T's files have a dependency on PreMxConfig.h and on PostMxConfig.h (see page 317) (through MxConfig.h), any modification done to these files requires the re-compilation of all the source code including MxConfig.h.

See Also

PostMxConfig.h (see page 317).

2.6.1.145 - PostMxConfig.h

Header file that can be included at the end of the MxConfig header file.

Description

The PostMxConfig header file can be used to add information required after the inclusion of MxConfig, or to override information given by PreMxConfig or MxConfig itself. An example of use would be to add an application-specific preprocessor macro based on the information generated by MxConfig (architecture, OS version, etc.). This file is automatically included by MxConfig.h when MXD_POST_CONFIG (see page 303) is defined.

See Also

MXD_POST_CONFIG (see page 303).

2.6.1.146 - MXD_OS_NONE Macro

Indicates that build is done for an unspecified operating system.

C++

```
#define MXD_OS_NONE
```

Description

Preprocessor macro used to disable operating system specific code. MXD_OS_XXX macros are mutually exclusive. This allows to have a minimal port of the framework without most of its operating system abstractions like threads, timers, mutexes, and semaphores. When defined, the support for native 64 bits cannot be deduced.

Location

Define this in "PreMxConfig.h".

See Also

MXD_64BITS_SUPPORT_DISABLE (see page 268), MXD_64BITS_CUSTOM_TYPE (see page 268),
 MXD_TRACE_FORMAT_HANDLER_OVERRIDE (see page 310), MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE (see page 309).

2.6.2 - Variables

This section documents the variables of the Sources/Config folder.

Macros

Macro	Description
szVERSION_FW (see page 318)	The current version of the framework.

2.6.2.1 - szVERSION_FW Macro

The current version of the framework.

C++

```
#define szVERSION_FW "2.1.8.17"
```

Description

This string contains the current framework version number.

Location

Defined in Config/VersionMitosFw.h.

2.7 - Crypto

This section documents the Sources/Crypto folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 318)
- Enumerations (see page 409)
- Variables (see page 411)

2.7.1 - Classes

This section documents the classes of the Sources/Crypto folder.

Classes

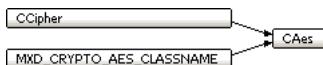
Class	Description
CAes (see page 319)	Class that manages the AES algorithm.
CBase64 (see page 326)	Class that manages Base64 encoding.
CCipher (see page 331)	Base class for all cipher algorithms.
CCrc (see page 336)	Used to calculate a cyclic redundancy check (CRC) on an array of bytes.
CCrypto (see page 339)	Used to initialize the crypto library.
CDiffieHellman (see page 340)	Class that manages the Diffie-Hellman algorithm.
CHash (see page 351)	Base class for all hash algorithms.
CHMac (see page 355)	Base class for all HMAC algorithms.

CMd5 (see page 360)	Class that manages the MD5 algorithm.
CMd5Mac (see page 364)	Class that manages the MD5 message authentication code algorithm.
CPrivateKey (see page 368)	Class that manages the private key for all algorithms.
CPublicKey (see page 374)	Class that manages the public key for all algorithms.
CRsa (see page 379)	Class that manages the RSA algorithm.
CSecurePrng (see page 389)	Class implementing a secure pseudo-random generator.
CSecureSeed (see page 390)	Class used to generate random bytes to seed a pseudo-random generator.
CSha1 (see page 391)	Class that manages the SHA-1 algorithm.
CSha1Mac (see page 395)	Class that manages the SHA-1 Message Authentication Code algorithm.
CSha2 (see page 399)	Class that manages the SHA-2 algorithm.
CSha2Mac (see page 404)	Class that manages the SHA-2 Message Authentication Code algorithm.
IPassPhrase (see page 408)	Interface through which requests for pass phrases are done to decrypt an object stored in PEM format.

2.7.1.1 - CAes Class

Class that manages the AES algorithm.

Class Hierarchy



C++

```
class CAes : public CCipher, public MXD_CRYPTO_AES_CLASSNAME;
```

Description

CAes is the class that manages the AES algorithm. The algorithm decrypts/encrypts data by blocks of 16 bytes. If a block of less than 16 bytes is passed to the Update (see page 325) method, this block is temporarily stored internally until new data is added and the block is completed or the End (see page 321) method is called.

Location

Crypto/CAes.h

Constructors

Constructor	Description
CAes (see page 320)	Constructor.

CCipher Class

CCipher Class	Description
CCipher (see page 332)	Constructor.

Legend



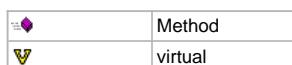
Destructors

Destructor	Description
~CAes (see page 320)	Destructor.

CCipher Class

CCipher Class	Description
~CCipher (see page 332)	Destructor.

Legend



Methods

Method	Description
Begin (see page 320)	Initiates the cipher prior to its utilization.
End (see page 321)	Decrypts or Encrypts the rest of the data.
GetAlgorithm (see page 322)	Returns the underlying algorithm used by the cipher.

• GetBlockSizeInBits (see page 322)	Returns the size of each block in bits.
• GetBlockSizeInByte (see page 322)	Returns the size of each block in bytes.
• GetSupportedModes (see page 322)	Returns the modes supported by the cipher.
• SetDefaultAction (see page 323)	Sets the default action used.
• SetDefaultIV (see page 323)	Sets the default initialization vector.
• SetDefaultKey (see page 324)	Sets the default key.
• SetDefaultMode (see page 325)	Sets the default mode used.
• Update (see page 325)	Decrypts or encrypts input data.

CCipher Class

CCipher Class	Description
• Begin (see page 333)	Initiates the cipher prior to its utilization.
• End (see page 333)	Decrypts or Encrypts the rest of the data.
• GetAlgorithm (see page 334)	Returns the underlying algorithm used by the cipher.
• GetBlockSizeInBits (see page 334)	Returns the size of each block in bits.
• GetBlockSizeInByte (see page 334)	Returns the size of each block in bytes.
• GetSupportedModes (see page 335)	Returns the modes supported by the cipher.
• Update (see page 335)	Decrypts or encrypts input data.

Legend

	Method
	virtual
	abstract

2.7.1.1.1 - Constructors

2.7.1.1.1.1 - CAes::CAes Constructor

Constructor.

C++

```
CAes();
```

Description

Basic constructor for the object.

2.7.1.1.2 - Destructors

2.7.1.1.2.1 - CAes::~CAes Destructor

Destructor.

C++

```
virtual ~CAes();
```

Description

Destructor for the object.

2.7.1.1.3 - Methods

2.7.1.1.3.1 - Begin

2.7.1.1.3.1.1 - CAes::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const CBlob* pKey, IN const CBlob* pIV = NULL);
```

Parameters

Parameters	Description
eAction	Action to perform.
eMode	The mode in which to use the cipher. If the cipher does support multiple modes, then eMODE_DEFAULT MUST be specified.
pKey	Pointer containing the key blob to use for encryption/decryption. It may be NULL if the cipher does not require a key.
pIV	Pointer containing an initialization vector blob. It may be NULL if the cipher does not require an initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.1.3.1.2 - CAes::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const uint8_t* puKey, IN unsigned int uKeySize,
IN const uint8_t* puIV = NULL, IN unsigned int uIVSize = 0);
```

Parameters

Parameters	Description
eAction	Action to perform.
eMode	The mode in which to use the cipher. If the cipher does not have multiple modes, then eMODE_DEFAULT MUST be specified.
puKey	Pointer containing the key to encrypt/decrypt. It may be NULL if the cipher does not require a key.
uKeySize	Size of the key in bytes.
puIV	Pointer containing an initialization vector. It may be NULL if the cipher does not require an initialization vector.
uIVSize	Size of the initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.1.3.2 - End

2.7.1.1.3.2.1 - CAes::End Method

Decrypts or Encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT CBlob* pOutData, IN bool bAppend = true);
```

Parameters

Parameters	Description
puOutData	Pointer to contain the last data block.
bAppend	True if data is to be appended to the end of the blob, false otherwise.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer. If append is true, the data is appended to the end of the blob.

2.7.1.1.3.2.2 - CAes::End Method

Decrypts or encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT uint8_t* puOutData, OUT unsigned int* puOutDataSize);
```

Parameters

Parameters	Description
puOutData	Pointer to contain the last data block.
puOutDataSize	Size of the output data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer.

2.7.1.1.3.3 - CAes::GetAlgorithm Method

Returns the underlying algorithm used by the cipher.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

EAlgorithm.

Description

Returns the underlying algorithm used by the cipher.

2.7.1.1.3.4 - CAes::GetBlockSizeInBits Method

Returns the size of each block in bits.

C++

```
virtual unsigned int GetBlockSizeInBits();
```

Returns

Size of a block in bits. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bits.

2.7.1.1.3.5 - CAes::GetBlockSizeInByte Method

Returns the size of each block in bytes.

C++

```
virtual unsigned int GetBlockSizeInByte();
```

Returns

Size of a block in bytes. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bytes.

2.7.1.1.3.6 - CAes::GetSupportedModes Method

Returns the modes supported by the cipher.

C++

```
virtual const CVector<EMode>* GetSupportedModes();
```

Returns

CVector containing EModes.

Description

Returns the modes supported by the cipher.

2.7.1.1.3.7 - CAes::SetDefaultAction Method

Sets the default action used.

C++

```
mxt_result SetDefaultAction(IN EAction eAction);
```

Parameters

Parameters	Description
IN EAction eAction	Action to use.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default action to use when eACTION_DEFAULT is passed to the Begin (see page 320) method.

See Also

Begin (see page 320)

2.7.1.1.3.8 - SetDefaultIV**2.7.1.1.3.8.1 - CAes::SetDefaultIV Method**

Sets the default initialization vector.

C++

```
mxt_result SetDefaultIV(IN const CBlob* pIV);
```

Parameters

Parameters	Description
IN const CBlob* pIV	Pointer to the blob containing the initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default initialization vector. This parameter is used when the initialization vector is not specified in the Begin (see page 320) method.

See Also

Begin (see page 320)

2.7.1.1.3.8.2 - CAes::SetDefaultIV Method

Sets the default IV.

C++

```
mxt_result SetDefaultIV(IN const uint8_t* puIV, IN unsigned int uIVSize);
```

Parameters

Parameters	Description
IN const uint8_t* puIV	Pointer to the initialization vector.
IN unsigned int uIVSize	Size of the initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default initialization vector. This parameter is used when the initialization vector is not specified in the Begin (see page 320) method.

See Also

Begin (see page 320)

2.7.1.1.3.9 - SetDefaultKey**2.7.1.1.3.9.1 - CAes::SetDefaultKey Method**

Sets the default key.

C++

```
mxt_result SetDefaultKey(IN const CBlob* pKey);
```

Parameters

Parameters	Description
pKeyV	Pointer to the blob containing the key.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default key. This parameter is used when the key is not specified in the Begin (see page 320) method.

See Also

Begin (see page 320)

2.7.1.1.3.9.2 - CAes::SetDefaultKey Method

Sets the default key.

C++

```
mxt_result SetDefaultKey(IN const uint8_t* puKey, IN unsigned int uKeySize);
```

Parameters

Parameters	Description
puKeyV	Pointer to the key.
uIVSize	Size of the key.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default key. This parameter is used when the key is not specified (= NULL) in the Begin (see page 320) method. When used, it increases the performance of the "Begin (see page 320)" method because the key is not set in the AES algorithm each time it is called. The key is rather set only once until a new key is set or a key is used in the "Begin (see page 320)" method.

See Also

Begin (see page 320)

2.7.1.1.3.10 - CAes::SetDefaultMode Method

Sets the default mode used.

C++

```
mxt_result SetDefaultMode(IN EMode eMode);
```

Parameters

Parameters	Description
IN EMode eMode	Mode to use

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Sets the default mode to use when eMODE_DEFAULT is passed to the Begin (see page 320) method.

See Also

Begin (see page 320)

2.7.1.1.3.11 - Update

2.7.1.1.3.11.1 - CAes::Update Method

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const CBlob* pInData, OUT CBlob* pOutData, IN bool bAppend = true);
```

Parameters

Parameters	Description
pInData	Pointer to the blob to encrypt.
pOutData	Pointer to a blob to contain the complete decrypted/encrypted blocks.
bAppend	True if data is to be appended to the end of the blob, false otherwise.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts pInData->Size() bytes from pInData and writes the result to pOutData. The resulting amount of data depends on the algorithm used. If append is true, the data is appended to the end of the blob.

2.7.1.1.3.11.2 - CAes::Update Method

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const uint8_t* puInData, IN unsigned int uInDataSize, OUT uint8_t* puOutData, OUT
unsigned int* puOutDataSize);
```

Parameters

Parameters	Description
puInData	Pointer to the byte array to decrypt/encrypt.
uInDataSize	Size of the byte array to decrypt/encrypt.
puOutData	Pointer to contain the complete decrypted/encrypted blocks.
puOutDataSize	Size of the outputted data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts `ulnDataSize` bytes from the buffer `puInData` and writes the result to `puOutData`. The resulting amount of data depends on the algorithm used. Sufficient space **MUST** be available. The amount of encrypted data returned depends on the number of complete cipher blocks that are available. Any leftover data is kept and used in future Update calls or with the End method.

See Also

End

2.7.1.2 - CBase64 Class

Class that manages Base64 encoding.

Class Hierarchy



C++

```
class CBase64 : public CCipher;
```

Description

`CBase64` is the class that manages Base64 encoding. Base64 encoding takes groups of 24 bits with each character encoded in 8 bits and transforms it to 4 characters each encoded in 6 bits. Each of these characters is then mapped to one of 64 possible characters. If there are not enough characters to complete the encoding, a 65th character, '=', is appended at the end.

To decode, each series of four 6 bit characters is converted back into 3 8 bit characters.

The block size of the input buffer is 3 bytes long and the size of an output block is 4 bytes long.

To get the size of the output buffer, it is possible to call the update method with the output buffer equal to `NULL`. The buffer can then be allocated with the appropriate size to contain the output.

Location

Crypto/CBase64.h

Constructors

Constructor	Description
<code>CBase64</code> (see page 327)	Constructor.

CCipher Class

CCipher Class	Description
<code>CCipher</code> (see page 332)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
<code>~CBase64</code> (see page 327)	Destructor.

CCipher Class

CCipher Class	Description
<code>~CCipher</code> (see page 332)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
<code>Begin</code> (see page 327)	Initiates the cipher prior to its utilization.
<code>End</code> (see page 328)	Decrypts or Encrypts the rest of the data.
<code>GetAlgorithm</code> (see page 329)	Returns the underlying algorithm used by the cipher.
<code>GetBlockSizeInBits</code> (see page 329)	Returns the size of each block in bits.

◆VV GetBlockSizeInByte (see page 329)	Returns the size of each block in bytes.
◆VV GetSupportedModes (see page 329)	Returns the modes supported by the cipher.
◆ GetVariant (see page 330)	Gets the variant.
◆ SetVariant (see page 330)	Sets the variant.
◆VV Update (see page 330)	Decrypts or encrypts input data.

CCipher Class

CCipher Class	Description
◆ A Begin (see page 333)	Initiates the cipher prior to its utilization.
◆ A End (see page 333)	Decrypts or Encrypts the rest of the data.
◆ A GetAlgorithm (see page 334)	Returns the underlying algorithm used by the cipher.
◆ A GetBlockSizeInBits (see page 334)	Returns the size of each block in bits.
◆ A GetBlockSizeInByte (see page 334)	Returns the size of each block in bytes.
◆ A GetSupportedModes (see page 335)	Returns the modes supported by the cipher.
◆ A Update (see page 335)	Decrypts or encrypts input data.

Legend

◆	Method
V	virtual
A	abstract

2.7.1.2.1 - Constructors

2.7.1.2.1.1 - CBase64::CBase64 Constructor

Constructor.

C++

```
CBase64();
```

Description

Basic constructor for the object.

2.7.1.2.2 - Destructors

2.7.1.2.2.1 - CBase64::~CBase64 Destructor

Destructor.

C++

```
virtual ~CBase64();
```

Description

Destructor for the object.

2.7.1.2.3 - Methods

2.7.1.2.3.1 - Begin

2.7.1.2.3.1.1 - CBase64::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const CBlob* pblobKey, IN const CBlob* pblobIV = NULL);
```

Parameters

Parameters	Description
eAction	Action to perform.
eMode	The mode in which to use the cipher. If the cipher does support multiple modes, then eMODE_DEFAULT MUST be specified.
pKey	Pointer containing the key blob to use for encryption/decryption. It may be NULL if the cipher does not require a key.
pIV	Pointer containing an initialization vector blob. It may be NULL if the cipher does not require an initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.2.3.1.2 - CBase64::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const uint8_t* puKey = NULL, IN unsigned int uKeySize = 0, IN const uint8_t* puIV = NULL, IN unsigned int uIVSize = 0);
```

Parameters

Parameters	Description
eAction	Action to perform.
eMode	The mode in which to use the cipher. If the cipher does not have multiple modes, then eMODE_DEFAULT MUST be specified.
puKey	Pointer containing the key to encrypt/decrypt. It may be NULL if the cipher does not require a key.
uKeySize	Size of the key in bytes.
puIV	Pointer containing an initialization vector. It may be NULL if the cipher does not require an initialization vector.
uIVSize	Size of the initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.2.3.2 - End

2.7.1.2.3.2.1 - CBase64::End Method

Decrypts or Encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT CBlob* pblobOutData, IN bool bAppend = true);
```

Parameters

Parameters	Description
pblobOutData	Pointer to contain the last data block.
bAppend	True if data is to be appended to the end of the blob, false otherwise.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer. If append is true, the data is appended to the end of the blob.

2.7.1.2.3.2.2 - CBase64::End Method

Decrypts or encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT uint8_t* puOutData, OUT unsigned int* puOutDataSize);
```

Parameters

Parameters	Description
puOutData	Pointer to contain the last data block.
puOutDataSize	Size of the output data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer.

2.7.1.2.3.3 - CBase64::GetAlgorithm Method

Returns the underlying algorithm used by the cipher.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

EAlgorithm.

Description

Returns the underlying algorithm used by the cipher.

2.7.1.2.3.4 - CBase64::GetBlockSizeInBits Method

Returns the size of each block in bits.

C++

```
virtual unsigned int GetBlockSizeInBits();
```

Returns

Size of a block in bits. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bits.

2.7.1.2.3.5 - CBase64::GetBlockSizeInByte Method

Returns the size of each block in bytes.

C++

```
virtual unsigned int GetBlockSizeInByte();
```

Returns

Size of a block in bytes. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bytes.

2.7.1.2.3.6 - CBase64::GetSupportedModes Method

Returns the modes supported by the cipher.

C++

```
virtual const CVector<EMode>* GetSupportedModes();
```

Returns

CVector containing EModes.

Description

Returns the modes supported by the cipher.

2.7.1.2.3.7 - CBase64::GetVariant Method

Gets the variant.

C++

```
EVariant GetVariant() const;
```

Returns

The variant of the CBase64 (see page 326) object.

Description

Gets the variant of a CBase64 (see page 326).

2.7.1.2.3.8 - CBase64::SetVariant Method

Sets the variant.

C++

```
void SetVariant(EVariant eVariant);
```

2.7.1.2.3.9 - Update**2.7.1.2.3.9.1 - CBase64::Update Method**

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const CBlob* pblobInData, OUT CBlob* pblobOutData, IN bool bAppend = true);
```

Parameters

Parameters	Description
pInData	Pointer to the blob to encrypt.
pOutData	Pointer to a blob to contain the complete decrypted/encrypted blocks.
bAppend	True if data is to be appended to the end of the blob, false otherwise.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts pInData->Size() bytes from pInData and writes the result to pOutData. The resulting amount of data depends on the algorithm used. If append is true, the data is appended to the end of the blob.

2.7.1.2.3.9.2 - CBase64::Update Method

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const uint8_t* puInData, IN unsigned int uInDataSize, OUT uint8_t* puOutData, OUT unsigned int* puOutDataSize);
```

Parameters

Parameters	Description
puInData	Pointer to the byte array to decrypt/encrypt.
uInDataSize	Size of the byte array to decrypt/encrypt.
puOutData	Pointer to contain the complete decrypted/encrypted blocks.
puOutDataSize	Size of the outputted data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts uInDataSize bytes from the buffer puInData and writes the result to puOutData. The resulting amount of data depends on the algorithm used. Sufficient space MUST be available. The amount of encrypted data returned depends on the number of complete cipher blocks that are available. Any leftover data is kept and used in future Update calls or with the End method.

See Also

End

2.7.1.3 - CCipher Class

Base class for all cipher algorithms.

Class Hierarchy

CCipher

C++

```
class CCipher;
```

Description

CCipher is the base class for all cipher algorithms.

There are two types of cipher algorithms, stream cipher and block cipher. Both are managed by CCipher. A stream cipher is an algorithm that encrypts data on a byte per byte basis. A block cipher operates on fixed length block size, such as 16 bytes in the case of AES. Encryption is performed on a block by block basis. If longer messages are to be encrypted, several modes of operation may be used.

The simplest mode of operation is Electronic Cookbook (ECB), in which the message is simply split into blocks and each block is encrypted. This makes identical blocks to give identical cipher blocks after encryption and does not hide data patterns. Padding is required with this mode.

The Cipher-Block Chain (CBC) mode XORs each block with the previous cipher-text block. Each cipher-text block thus depends on all the plain text blocks up to that point. Padding is required with this mode.

Counter (CTR) Cipher Feedback (CFB) and Output Feedback (OFB) turn the block cipher into a stream cipher. They do not require padding. They generate key-stream blocks that are then XORed with the plaintext to give the cipher text. CTR generates the next key-stream by encrypting successive values of a counter. CFB generates the next key-stream by encrypting the previous cipher-text block. OFB generates the key-stream by encrypting the previous key-stream block.

The ECB and CBC modes require the input to be an exact multiple of their block size (sixteen bytes for AES). If the plaintext is not a multiple of block size, it needs to be padded. With AES, the End (see page 333) method returns an error if the leftover number of bytes from the Update (see page 335) method is not a multiple of sixteen. The receiving side needs to know how to remove the padding.

```
mxt_result PerformCipher(IN CCipher::EAction eAction,
                         IN CCipher::EMode eMode,
                         IN const CBlob* pKey,
                         IN const CBlob* pIV,
                         IN const CBlob* pInData,
                         OUT CBlob* pOutData,
                         IN CCipher* pCipher)
{
    mxt_result res = resS_OK;

    res = pCipher->Begin(eAction, eMode, pKey, pIV);
    if (MX_RIS_S(res))
    {
        res = pCipher->Update(pInData, pOutData);
    }
    if (MX_RIS_S(res))
    {
        res = pCipher->End();
    }
}
```

```

        res = pCipher->End(pOutData, true);
    }

    return res;
}

```

Location

Crypto/CCipher.h

Constructors

Constructor	Description
 CCipher (see page 332)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CCipher (see page 332)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
  Begin (see page 333)	Initiates the cipher prior to its utilization.
  End (see page 333)	Decrypts or Encrypts the rest of the data.
  GetAlgorithm (see page 334)	Returns the underlying algorithm used by the cipher.
  GetBlockSizeInBits (see page 334)	Returns the size of each block in bits.
  GetBlockSizeInByte (see page 334)	Returns the size of each block in bytes.
  GetSupportedModes (see page 335)	Returns the modes supported by the cipher.
  Update (see page 335)	Decrypts or encrypts input data.

Legend

	Method
	abstract

2.7.1.3.1 - Constructors**2.7.1.3.1.1 - CCipher::CCipher Constructor**

Constructor.

C++`CCipher();`**Description**

Basic constructor.

2.7.1.3.2 - Destructors**2.7.1.3.2.1 - CCipher::~CCipher Destructor**

Destructor.

C++`virtual ~CCipher();`**Description**

Destructor.

2.7.1.3.3 - Methods

2.7.1.3.3.1 - Begin

2.7.1.3.3.1.1 - CCipher::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const CBlob* pKey, IN const CBlob* pIV) = 0;
```

Parameters

Parameters	Description
IN EAction eAction	Action to perform.
IN EMode eMode	The mode in which to use the cipher. If the cipher does support multiple modes, then eMODE_DEFAULT MUST be specified.
IN const CBlob* pKey	Pointer containing the key blob to use for encryption/decryption. It may be NULL if the cipher does not require a key.
IN const CBlob* pIV	Pointer containing an initialization vector blob. It may be NULL if the cipher does not require an initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.3.3.1.2 - CCipher::Begin Method

Initiates the cipher prior to its utilization.

C++

```
virtual mxt_result Begin(IN EAction eAction, IN EMode eMode, IN const uint8_t* puKey = NULL, IN unsigned int uKeySize = 0, IN const uint8_t* puIV = NULL, IN unsigned int uIVSize = 0) = 0;
```

Parameters

Parameters	Description
IN EAction eAction	Action to perform.
IN EMode eMode	The mode in which to use the cipher. If the cipher does not have multiple modes, then eMODE_DEFAULT MUST be specified.
IN const uint8_t* puKey = NULL	Pointer containing the key to encrypt/decrypt. It may be NULL if the cipher does not require a key.
IN unsigned int uKeySize = 0	Size of the key in bytes.
IN const uint8_t* puIV = NULL	Pointer containing an initialization vector. It may be NULL if the cipher does not require an initialization vector.
IN unsigned int uIVSize = 0	Size of the initialization vector.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Initializes the cipher prior to its utilization. Specifies the action to be performed, the cipher mode, the key, and the initialization vector.

2.7.1.3.3.2 - End

2.7.1.3.3.2.1 - CCipher::End Method

Decrypts or Encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT CBlob* pOutData, IN bool bAppend = true) = 0;
```

Parameters

Parameters	Description
IN bool bAppend = true	True if data is to be appended to the end of the blob, false otherwise.
puOutData	Pointer to contain the last data block.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer. If append is true, the data is appended to the end of the blob.

2.7.1.3.3.2.2 - CCipher::End Method

Decrypts or encrypts the rest of the data.

C++

```
virtual mxt_result End(OUT uint8_t* puOutData, OUT unsigned int* puOutDataSize) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puOutData	Pointer to contain the last data block.
OUT unsigned int* puOutDataSize	Size of the output data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts the rest of the data that is left within the cipher and writes it to the output buffer.

2.7.1.3.3.3 - CCipher::GetAlgorithm Method

Returns the underlying algorithm used by the cipher.

C++

```
virtual EAlgorithm GetAlgorithm() = 0;
```

Returns

EAlgorithm (see page 410).

Description

Returns the underlying algorithm used by the cipher.

2.7.1.3.3.4 - CCipher::GetBlockSizeInBits Method

Returns the size of each block in bits.

C++

```
virtual unsigned int GetBlockSizeInBits() = 0;
```

Returns

Size of a block in bits. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bits.

2.7.1.3.3.5 - CCipher::GetBlockSizeInByte Method

Returns the size of each block in bytes.

C++

```
virtual unsigned int GetBlockSizeInByte() = 0;
```

Returns

Size of a block in bytes. Returns 0 if no block size is available for the specified algorithm.

Description

Returns the size of each block in bytes.

2.7.1.3.3.6 - CCipher::GetSupportedModes Method

Returns the modes supported by the cipher.

C++

```
virtual const CVector<EMode>* GetSupportedModes() = 0;
```

Returns

CVector (See page 227) containing EModes.

Description

Returns the modes supported by the cipher.

2.7.1.3.3.7 - Update**2.7.1.3.3.7.1 - CCipher::Update Method**

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const CBlob* pInData, OUT CBlob* pOutData, IN bool bAppend = true) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pInData	Pointer to the blob to encrypt.
OUT CBlob* pOutData	Pointer to a blob to contain the complete decrypted/encrypted blocks.
IN bool bAppend = true	True if data is to be appended to the end of the blob, false otherwise.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts pInData->Size() bytes from pInData and writes the result to pOutData. The resulting amount of data depends on the algorithm used. If append is true, the data is appended to the end of the blob.

2.7.1.3.3.7.2 - CCipher::Update Method

Decrypts or encrypts input data.

C++

```
virtual mxt_result Update(IN const uint8_t* puInData, IN unsigned int uInDataSize, OUT uint8_t* puOutData, OUT unsigned int* puOutDataSize) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puInData	Pointer to the byte array to decrypt/encrypt.
IN unsigned int uInDataSize	Size of the byte array to decrypt/encrypt.
OUT uint8_t* puOutData	Pointer to contain the complete decrypted/encrypted blocks.
OUT unsigned int* puOutDataSize	Size of the outputted data.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

Decrypts or encrypts uInDataSize bytes from the buffer puInData and writes the result to puOutData. The resulting amount of data depends on the algorithm used. Sufficient space MUST be available. The amount of encrypted data returned depends on the number of complete cipher blocks that are available. Any leftover data is kept and used in future Update calls or with the End (see page 333) method.

See Also

End (see page 333)

2.7.1.4 - CCrc Class

Used to calculate a cyclic redundancy check (CRC) on an array of bytes.

Class Hierarchy

CCrc

C++

```
class CCrc;
```

Description

This class is used to calculate a cyclic redundancy check (CRC) on an array of bytes.

The CRC check is initialized by a call to Begin (see page 337). Begin (see page 337) takes multiple types of CRC checks that can be made on the array(s). One or many byte arrays are then passed to Update (see page 338) and the CRC is calculated for every byte array received. Once all arrays have been passed, the complement (XOR) of the CRC is returned by calling End (see page 337). The complement is returned since this is the value that must be appended to the byte array for FCS verification.

CRC Verification: To check if an array is correct, it must have the complement (XOR) of its CRC appended to the byte array. Verification is done by the algorithm on the array with the complemented bytes appended to the end. The new CRC should give a specific value: 0xf0b8 for CRC16 CCITT and 0xdeb20e3 for CRC32 IEEE 802.3. No value for CRC32C has been found in the document related to the algorithm. If the value of the new CRC is different from these values, then there has been an error in the transmission of data. For this implementation, the complement (XOR) of the returned CRC must be compared since the complement is already returned by End (see page 337).

Location

Crypto/CCrc.h

Constructors

Constructor	Description
CCrc (see page 337)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CCrc (see page 337)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 Begin (see page 337)	Initializes the CRC calculation for data blocks.
 End (see page 337)	Returns the value of the current 16 bits CRC.
 Update (see page 338)	

Legend

	Method
---	--------

2.7.1.4.1 - Constructors

2.7.1.4.1.1 - CCrc::CCrc Constructor

Constructor.

C++

```
CCrc();
```

Description

Default constructor.

2.7.1.4.2 - Destructors

2.7.1.4.2.1 - CCrc::~CCrc Destructor

Destructor.

C++

```
virtual ~CCrc();
```

Description

Default destructor.

2.7.1.4.3 - Methods

2.7.1.4.3.1 - CCrc::Begin Method

Initializes the CRC calculation for data blocks.

C++

```
mxt_result Begin(IN ECrcType eCrcType);
```

Parameters

Parameters	Description
IN ECrcType eCrcType	Type of CRC to use.

Returns

resS_OK if successful, resFE_INVALID_STATE otherwise.

Description

Initializes the default value for the CRC calculation and the type of CRC to be performed during calls to update.

See Also

Update (see page 338), End (see page 337)

2.7.1.4.3.2 - End

2.7.1.4.3.2.1 - CCrc::End Method

Returns the value of the current 16 bits CRC.

C++

```
mxt_result End(OUT uint16_t* puCrc16);
```

Parameters

Parameters	Description
OUT uint16_t* puCrc16	Pointer to a 16 bits variable.

Returns

resS_OK if successful, resFE_INVALID_STATE, or resFE_INVALID_ARGUMENT otherwise.

Description

Copies the value of the complement of the CRC into a 16 bits variable.

See Also

Begin (see page 337), Update (see page 338)

2.7.1.4.3.2.2 - CCrc::End Method

Returns the value of the current 32 bits CRC.

C++

```
mxt_result End(OUT uint32_t* puCrc32);
```

Parameters

Parameters	Description
OUT uint32_t* puCrc32	Pointer to a 32 bits variable.

Returns

resS_OK if successful, resFE_INVALID_STATE, or resFE_INVALID_ARGUMENT otherwise.

Description

Copies the value of the complement of the CRC into a 32 bits variable.

See Also

Begin (see page 337), Update (see page 338)

2.7.1.4.3.3 - Update**2.7.1.4.3.3.1 - CCrc::Update Method**

```
mxt_result Update(IN CBlob* pblobData);
```

Parameters

Parameters	Description
IN CBlob* pblobData	Pointer to a CBlob (see page 95) that contains the data to calculate the CRC.

Returns

resS_OK if successful, resFE_FAIL, resFE_INVALID_STATE or, resFE_INVALID_ARGUMENT otherwise.

Description

Calculates the CRC with the current byte array. This value is appended to any prior CRC calculated. It automatically uses the proper algorithm for either 16 bits CRC or 32 bits CRC.

See Also

Begin (see page 337), End (see page 337)

2.7.1.4.3.3.2 - CCrc::Update Method

Inserts a data block from which to calculate the CRC.

C++

```
mxt_result Update(IN const uint8_t* puData, unsigned int uDataSize);
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to a byte array that contains the data to calculate the CRC.
unsigned int uDataSize	Size of the byte array.

Returns

resS_OK if successful, resFE_FAIL, resFE_INVALID_STATE or, resFE_INVALID_ARGUMENT otherwise.

Description

Calculates the CRC with the current byte array. This value is appended to any prior CRC calculated. It uses the algorithm for either 16 bits CRC or 32 bits CRC as defined in Begin (see page 337).

See Also

Begin (see page 337), End (see page 337)

2.7.1.5 - CCrypto Class

Used to initialize the crypto library.

Class Hierarchy

 CCrypto

C++

```
class CCrypto;
```

Description

CCrypto is a class that is only used to initialize the crypto library.

Location

Crypto/CCrypto.h

Methods

Method	Description
◆ Enter (see page 339)	Enters the concurrency protection domain.
◆ Exit (see page 339)	Exits the concurrency protection domain.
◆ Instance (see page 340)	Returns the unique instance of CCrypto
◆ MocanaLog (see page 340)	Mocana SS tracing callback.

Legend

 Method

2.7.1.5.1 - Methods

2.7.1.5.1.1 - CCrypto::Enter Method

Enters the concurrency protection domain.

C++

```
void Enter();
```

Description

Enters the concurrency protection domain. Note that this method reserves the system wide crypto critical section. Exit (see page 339) from it quickly by using the method Exit (see page 339).

See Also

Exit (see page 339)

2.7.1.5.1.2 - CCrypto::Exit Method

Exits the concurrency protection domain.

C++

```
void Exit();
```

Description

Exits the concurrency protection domain.

See Also

Enter (see page 339)

2.7.1.5.1.3 - CCrypto::Instance Method

Returns the unique instance of CCrypto (see page 339)

C++

```
static CCrypto* Instance();
```

Returns

Pointer to the unique CCrypto (see page 339) instance.

Description

Returns the unique instance of the CCrypto (see page 339) class.

2.7.1.5.1.4 - CCrypto::MocanaLog Method

Mocana SS tracing callback.

C++

```
static void MocanaLog(int nModule, int nLevel, int8_t* szMsg);
```

Parameters

Parameters	Description
int nModule	The Mocana module.
int nLevel	The Mocana severity rating of the trace.
int8_t* szMsg	The message to display.

Description

Mocana SS tracing callback. Mocana modules are mapped to strings and displayed and severity levels are mapped to an EMxTraceLevel (see page 16).

2.7.1.6 - CDiffieHellman Class

Class that manages the Diffie-Hellman algorithm.

Class Hierarchy



C++

```
class CDiffieHellman : public MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME;
```

Description

CDiffieHellman is the class that manages the Diffie-Hellman key exchange algorithm.

Diffie-Hellman:

The Diffie-Hellman algorithm is a key exchange protocol that was developed by Bailey Whitfield Diffie and Martin Hellman in 1976. It allows two users to exchange a secret key over an insecure medium without any prior exchange.

Suppose Alice and Bob want to agree on a shared secret key.

First, Alice must generate a random private key x and Bob generates a random private key y . Both x and y must meet certain mathematical criteria and be kept secret.

Second, they compute their public key using two parameters: a large prime p and a generator g , which they have both agreed upon beforehand. Alice computes $X = g^x \bmod p$ and Bob computes $Y = g^y \bmod p$.

Alice sends Bob her public key X and Bob sends Alice his public key Y .

Finally, Alice computes $k = (Y^x) \bmod p$, and Bob computes $k' = (X^y) \bmod p$. Both k and k' are equal to $g^{(x * y)} \bmod p$. Alice and Bob now both have a shared secret key k (k').

To successfully use Diffie-Hellman, a few steps must be executed in the correct order.

First, a generator g must be chosen. It affects the generation of the large prime p . These two parameters are public. They may be chosen by the initiator of the protocol or they may be hardcoded. The only rule is that the same parameters g and p must be used by both sides. To generate the prime p , the method GeneratePrime (see page 344) must be called. To use an already generated prime p , the method SetParameters (see page 350) must be called. It is important to note that the generation of the prime p is a very

CPU-intensive task.

Instead of choosing an arbitrary generator and generating a prime, Diffie-Hellman Oakley groups can be used. Groups 1 (768-bit prime), 2 (1024-bit prime), 5 (1536-bit prime), 14 (2048-bit prime), 15 (3072-bit prime), 16 (4096-bit prime), 17 (6144-bit prime) and 18 (8192-bit prime) are available in this class.

Once the parameters are known, the public and private keys must be generated. This is accomplished with a call to the method `GeneratePublicAndPrivateKeys` (see page 345).

Now, the initiator must send his public key. It is possible to also send a generator g and a prime p if required. All these values are public and may be observed without any security breach. Only the private key must remain secret.

Upon reception of the peer's public key, the initiator must call the method `GenerateSharedKey` (see page 345) using the peer's public key.

From now on, both sides have access to the same shared key.

<pre> Initiator g = generator GeneratePrime(1024, g) GeneratePublicAndPrivateKeys() p = GetPrime() X = GetPublicKey() </pre>	<pre> Acceptor SetParameters(p, g) GeneratePublicAndPrivateKeys() Y = GetPublicKey() </pre>
$\xleftarrow{\hspace{10em}} Y \xrightarrow{\hspace{10em}}$	
<pre> K = GenerateSharedKey(Y) </pre>	<pre> K = GenerateSharedKey(X) </pre>

Notes

Keys generated with Diffie-Hellman may not be of the exact length of the specified prime. The reason for this is that leading zeros in the keys are discarded. However, when retrieving the keys, they are appropriately padded.

Location

`Crypto/CDiffieHellman.h`

Constructors

Constructor	Description
 <code>CDiffieHellman</code> (see page 343)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 <code>~CDiffieHellman</code> (see page 344)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 <code>=</code> (see page 351)	Copies the referenced class into this one.

Legend

	Method
---	--------

Methods

Method	Description
 <code>GeneratePrime</code> (see page 344)	Generates a prime number.
 <code>GeneratePublicAndPrivateKeys</code> (see page 345)	Generates a private and a public key.
 <code>GenerateSharedKey</code> (see page 345)	Generates the shared key.
 <code>GetGenerator</code> (see page 346)	Gets the generator.
 <code>GetPrime</code> (see page 346)	Gets the prime number.
 <code>GetPrivateKey</code> (see page 347)	Gets the private key.
 <code>GetPublicKey</code> (see page 348)	Gets the public key.
 <code>GetSharedKey</code> (see page 349)	Gets the shared key.

 SetParameters (See page 350)

Sets the parameters used for generating keys.

Legend



2.7.1.6.1 - Data Members

2.7.1.6.1.1 - CDiffieHellman::ms_auOAKLEY1_PRIME Data Member

Oakley group 1 768-bit prime from RFC 2409: $2^{768} - 2^{704} - 1 + 2^{64} * ((2^{638} \pi) + 149686)$.

C++

```
const uint8_t ms_auOAKLEY1_PRIME[96];
```

2.7.1.6.1.2 - CDiffieHellman::ms_auOAKLEY14_PRIME Data Member

Oakley group 14 2048-bit prime from RFC 3526: $2^{2048} - 2^{1984} - 1 + 2^{64} * ((2^{1918} \pi) + 124476)$

C++

```
const uint8_t ms_auOAKLEY14_PRIME[256];
```

2.7.1.6.1.3 - CDiffieHellman::ms_auOAKLEY15_PRIME Data Member

Oakley group 15 3072-bit prime number from RFC 3526: $2^{3072} - 2^{3008} - 1 + 2^{64} * ((2^{2942} \pi) + 1690314)$

C++

```
const uint8_t ms_auOAKLEY15_PRIME[384];
```

2.7.1.6.1.4 - CDiffieHellman::ms_auOAKLEY16_PRIME Data Member

Oakley group 16 4096-bit prime number from RFC 3526: $2^{4096} - 2^{4032} - 1 + 2^{64} * ((2^{3966} \pi) + 240904)$

C++

```
const uint8_t ms_auOAKLEY16_PRIME[512];
```

2.7.1.6.1.5 - CDiffieHellman::ms_auOAKLEY17_PRIME Data Member

Oakley group 17 6144-bit prime number from RFC 3526: $2^{6144} - 2^{6080} - 1 + 2^{64} * ((2^{6014} \pi) + 929484)$

C++

```
const uint8_t ms_auOAKLEY17_PRIME[768];
```

2.7.1.6.1.6 - CDiffieHellman::ms_auOAKLEY18_PRIME Data Member

Oakley group 18 8192-bit prime number from RFC 3526: $2^{8192} - 2^{8128} - 1 + 2^{64} * ((2^{8062} \pi) + 4743158)$

C++

```
const uint8_t ms_auOAKLEY18_PRIME[1024];
```

2.7.1.6.1.7 - CDiffieHellman::ms_auOAKLEY2_PRIME Data Member

Oakley group 2 1024-bit prime from RFC 2409: $2^{1024} - 2^{960} - 1 + 2^{64} * ((2^{894} \pi) + 129093)$

C++

```
const uint8_t ms_auOAKLEY2_PRIME[128];
```

2.7.1.6.1.8 - CDiffieHellman::ms_auOAKLEY5_PRIME Data Member

Oakley group 5 1536-bit prime from RFC 3526: $2^{1536} - 2^{1472} - 1 + 2^{64} * ((2^{1406} \pi) + 741804)$

C++

```
const uint8_t ms_auOAKLEY5_PRIME[192];
```

2.7.1.6.1.9 - CDiffieHellman::ms_uOAKLEY1_GENERATOR Data Member

Oakley group 1 generator : 2.

C++

```
const unsigned int ms_uOAKLEY1_GENERATOR;
```

2.7.1.6.1.10 - CDiffieHellman::ms_uOAKLEY14_GENERATOR Data Member

Oakley group 14 generator: 2.

C++

```
const unsigned int ms_uOAKLEY14_GENERATOR;
```

2.7.1.6.1.11 - CDiffieHellman::ms_uOAKLEY15_GENERATOR Data Member

Oakley group 15 generator: 2.

C++

```
const unsigned int ms_uOAKLEY15_GENERATOR;
```

2.7.1.6.1.12 - CDiffieHellman::ms_uOAKLEY16_GENERATOR Data Member

Oakley group 16 generator: 2.

C++

```
const unsigned int ms_uOAKLEY16_GENERATOR;
```

2.7.1.6.1.13 - CDiffieHellman::ms_uOAKLEY17_GENERATOR Data Member

Oakley group 17 generator: 2.

C++

```
const unsigned int ms_uOAKLEY17_GENERATOR;
```

2.7.1.6.1.14 - CDiffieHellman::ms_uOAKLEY18_GENERATOR Data Member

Oakley group 18 generator: 2.

C++

```
const unsigned int ms_uOAKLEY18_GENERATOR;
```

2.7.1.6.1.15 - CDiffieHellman::ms_uOAKLEY2_GENERATOR Data Member

Oakley group 2 generator : 2.

C++

```
const unsigned int ms_uOAKLEY2_GENERATOR;
```

2.7.1.6.1.16 - CDiffieHellman::ms_uOAKLEY5_GENERATOR Data Member

Oakley group 5 generator: 2.

C++

```
const unsigned int ms_uOAKLEY5_GENERATOR;
```

2.7.1.6.2 - Constructors

2.7.1.6.2.1 - CDiffieHellman

2.7.1.6.2.1.1 - CDiffieHellman::CDiffieHellman Constructor

Default constructor.

C++

```
CDiffieHellman();
```

Description

Default constructor.

2.7.1.6.2.1.2 - CDiffieHellman::CDiffieHellman Constructor

Copy constructor.

C++

```
CDiffieHellman(IN const CDiffieHellman& rDiffieHellman);
```

Parameters

Parameters	Description
IN const CDiffieHellman& rDiffieHellman	Reference to a CDiffieHellman class.

Description

Copy constructor.

2.7.1.6.2.1.3 - CDiffieHellman::CDiffieHellman Constructor

Copy constructor.

C++

```
CDiffieHellman(IN const CDiffieHellman* pDiffieHellman);
```

Parameters

Parameters	Description
IN const CDiffieHellman* pDiffieHellman	Pointer to a CDiffieHellman class.

Description

Copy constructor.

2.7.1.6.3 - Destructors**2.7.1.6.3.1 - CDiffieHellman::~CDiffieHellman Destructor**

Destructor.

C++

```
virtual ~CDiffieHellman();
```

Description

Default destructor.

2.7.1.6.4 - Methods**2.7.1.6.4.1 - CDiffieHellman::GeneratePrime Method**

Generates a prime number.

C++

```
mxt_result GeneratePrime(IN unsigned int uPrimeSizeInBits = 1024, IN unsigned int uGenerator = 2);
```

Parameters

Parameters	Description
IN unsigned int uPrimeSizeInBits = 1024	Size of the prime number in bits. Default is 1024, in order to comply to the FIPS standard.
IN unsigned int uGenerator = 2	Value of the generator for the key. Default is 2.

Returns

- resS_OK: the prime has been successfully generated.
- resFE_FAIL: the prime generation failed.
- resFE_NOT_IMPLEMENTED: the prime generation is not supported in this configuration.

Description

Generates a prime p serving as the modulus in the modular exponentiation.

2.7.1.6.4.2 - CDiffieHellman::GeneratePublicAndPrivateKeys Method

Generates a private and a public key.

C++

```
mxt_result GeneratePublicAndPrivateKeys(IN unsigned int uPrivateKeySizeInBits = 0);
```

Parameters

Parameters	Description
IN unsigned int uPrivateKeySizeInBits = 0	Size of the private key in bits. Default is zero, meaning the private key will be the same size as the prime p .

Returns

- resS_OK: the keys have been successfully generated.
- resFE_FAIL: the generation failed or mandatory parameters are missing for the generation.

Description

Generates a random private key x , then a public key X , by applying the formula $X = g^x \bmod p$, using the generator g and the prime p .

Notes

As covered in [RES], section 6.16, smaller private keys may be used in some scenarios, without threatening the security of the protocol. Hence, when $uPrivateKeySizeInBits$ is non-zero, it will dictate the private key size.

See Also

[RES] E. Rescorla, "SSL and TLS (see page 794): Designing and Building Secure Systems", Addison-Wesley Professional, 2000, 499 pp.

2.7.1.6.4.3 - GenerateSharedKey**2.7.1.6.4.3.1 - CDiffieHellman::GenerateSharedKey Method**

Generates the shared key.

C++

```
mxt_result GenerateSharedKey(IN const CBlob* pblobRemotePublicKey);
```

Parameters

Parameters	Description
IN const CBlob* pblobRemotePublicKey	The remote public key (e.g. the key Y).

Returns

- resS_OK: the shared key has been successfully generated.

- resFE_FAIL: the generation failed.
- resFE_INVALID_ARGUMENT: either pblobRemotePublicKey is NULL.

Description

Let x be the private key, p the prime and Y the remote public key. It generates the shared key K, using the formula $K = Y^x \bmod p$.

2.7.1.6.4.3.2 - CDiffieHellman::GenerateSharedKey Method

Generates the shared key.

C++

```
mxt_result GenerateSharedKey(IN const uint8_t* puRemotePublicKey, IN unsigned int uRemotePublicKeySize);
```

Parameters

Parameters	Description
IN const uint8_t* puRemotePublicKey	The remote public key (e.g. the key Y).
IN unsigned int uRemotePublicKeySize	The size of the key in bytes (the actual byte array length).

Returns

- resS_OK: the shared key has been successfully generated.
- resFE_FAIL: the generation failed.
- resFE_INVALID_ARGUMENT: either puRemotePublicKey is NULL or uRemotePublicKeySize is zero.

Description

Let x be the private key, p the prime and Y the remote public key. It generates the shared key K, using the formula $K = Y^x \bmod p$.

2.7.1.6.4.4 - CDiffieHellman::GetGenerator Method

Gets the generator.

C++

```
mxt_result GetGenerator(OUT unsigned int* puGenerator) const;
```

Parameters

Parameters	Description
OUT unsigned int* puGenerator	unsigned integer to store the generator value.

Returns

- resS_OK: the generator has been retrieved.
- resFE_FAIL: the generator has not yet been set.
- resFE_INVALID_ARGUMENT: puGenerator is NULL.

Description

Retrieves the generator g.

2.7.1.6.4.5 - GetPrime

2.7.1.6.4.5.1 - CDiffieHellman::GetPrime Method

Gets the prime number.

C++

```
mxt_result GetPrime(IN unsigned int uMaxSize, OUT uint8_t* puPrime, OUT unsigned int* puPrimeSize) const;
```

Parameters

Parameters	Description
IN unsigned int uMaxSize	The maximum capacity of puPrime.
OUT uint8_t* puPrime	The array used to store the prime.
OUT unsigned int* puPrimeSize	The size of the prime in bytes.

Returns

- resS_OK: the prime has been retrieved.
- resFE_FAIL: the prime has not yet been set or generated.
- resFE_INVALID_ARGUMENT: puPrimeSize is NULL.

Description

Retrieves the prime p. If puPrime is NULL, the method returns the size of the prime in bytes in puPrimeSize.

2.7.1.6.4.5.2 - CDiffieHellman::GetPrime Method

Gets the prime number.

C++

```
mxt_result GetPrime(OUT CBlob* pblobPrime) const;
```

Returns

- resS_OK: the prime has been retrieved.
- resFE_FAIL: the prime has not yet been set or generated.
- resFE_INVALID_ARGUMENT: pblobPrime is NULL.

Description

Retrieves the prime p.

2.7.1.6.4.6 - GetPrivateKey**2.7.1.6.4.6.1 - CDiffieHellman::GetPrivateKey Method**

Gets the private key.

C++

```
mxt_result GetPrivateKey(IN unsigned int uMaxSize, OUT uint8_t* puPrivateKey, OUT unsigned int* puPrivateKeySize) const;
```

Parameters

Parameters	Description
IN unsigned int uMaxSize	The maximum capacity of puPublicKey.
OUT uint8_t* puPrivateKey	The array used to store the public key.
OUT unsigned int* puPrivateKeySize	The size of the public key in bytes.

Returns

- resS_OK: the private key has been retrieved.
- resFE_FAIL: the private key has not yet been set or generated.

- resFE_INVALID_ARGUMENT: puPrivateKeySize is NULL.

Description

Retrieves the private key used to generate the public key. If puPrivateKey is NULL, the method returns the size of the prime in bytes in puPrivateKeySize.

Notes

this key must remain secret to ensure the security of the protocol.

2.7.1.6.4.6.2 - CDiffieHellman::GetPrivateKey Method

Gets the private key.

C++

```
mxt_result GetPrivateKey(OUT CBlob* pblobPrivateKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobPrivateKey	The blob used to store the public key.

Returns

- resS_OK: the private key has been retrieved.
- resFE_FAIL: the private key has not yet been set or generated.
- resFE_INVALID_ARGUMENT: pblobPrivateKey is NULL.

Description

Retrieves the private key used to generate the public key.

Notes

this key must remain secret to ensure the security of the protocol.

2.7.1.6.4.7 - GetPublicKey

2.7.1.6.4.7.1 - CDiffieHellman::GetPublicKey Method

Gets the public key.

C++

```
mxt_result GetPublicKey(IN unsigned int uMaxSize, OUT uint8_t* puPublicKey, OUT unsigned int* puPublicKeySize) const;
```

Parameters

Parameters	Description
IN unsigned int uMaxSize	The maximum capacity of puPublicKey.
OUT uint8_t* puPublicKey	The array used to store the public key.
OUT unsigned int* puPublicKeySize	The size of the public key in bytes.

Returns

- resS_OK: the public key has been retrieved.
- resFE_FAIL: the public key has not yet been set or generated.
- resFE_INVALID_ARGUMENT: puPublicKeySize is NULL.

Description

Retrieves the public key that must be forwarded to the peer. If puPublicKey is NULL, the method returns the size of the prime in bytes in puPublicKeySize.

2.7.1.6.4.7.2 - CDiffieHellman::GetPublicKey Method

Gets the public key.

C++

```
mxt_result GetPublicKey(OUT CBlob* pblobPublicKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobPublicKey	The blob used to store the public key.

Returns

- resS_OK: the public key has been retrieved.
- resFE_FAIL: the public key has not yet been set or generated.
- resFE_INVALID_ARGUMENT: pblobPublicKey is NULL.

Description

Retrieves the public key that must be forwarded to the peer.

2.7.1.6.4.8 - GetSharedKey

2.7.1.6.4.8.1 - CDiffieHellman::GetSharedKey Method

Gets the shared key.

C++

```
mxt_result GetSharedKey(IN unsigned int uMaxSize, OUT uint8_t* puSharedKey, OUT unsigned int* puSharedKeySize) const;
```

Parameters

Parameters	Description
IN unsigned int uMaxSize	The maximum capacity of puSharedKey.
OUT uint8_t* puSharedKey	The array used to store the shared key.
OUT unsigned int* puSharedKeySize	The size of the shared key in bytes.

Returns

- resS_OK: the shared key has been retrieved.
- resFE_FAIL: the shared key has not yet been generated.
- resFE_INVALID_ARGUMENT: puSharedKeySize is NULL.

Description

Retrieves the shared key. If puSharedKey is NULL, the method returns the size of the prime in bytes in puPrimeSize.

2.7.1.6.4.8.2 - CDiffieHellman::GetSharedKey Method

Gets the shared key.

C++

```
mxt_result GetSharedKey(OUT CBlob* pblobSharedKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobSharedKey	The blob used to store the shared key.

Returns

- resS_OK: the shared key has been retrieved.
- resFE_FAIL: the shared key has not yet been generated.
- resFE_INVALID_ARGUMENT: pblobSharedKey is NULL.

Description

Retrieves the shared key.

2.7.1.6.4.9 - SetParameters

2.7.1.6.4.9.1 - CDiffieHellman::SetParameters Method

Sets the parameters used for generating keys.

C++

```
mxt_result SetParameters(IN const CBlob* pblobPrime, IN unsigned int uGenerator, IN const CBlob* pblobPublicKey,
IN const CBlob* pblobPrivateKey);
```

Parameters

Parameters	Description
IN const CBlob* pblobPrime	The prime modulus.
IN unsigned int uGenerator	The generator. It must be greater or equal to 2.
IN const CBlob* pblobPublicKey	The public key.
IN const CBlob* pblobPrivateKey	The private key.

Returns

- resS_OK: the parameters has successfully been set.
- resFE_FAIL: failed to set the parameters.
- resFE_INVALID_ARGUMENT: there is a missing mandatory parameter (depending on the context), or their values are invalid.

Description

Sets the prime, the generator, the public key, and the private key for use in the GeneratePublicAndPrivateKeys (see page 345) and GenerateSharedKey (see page 345) methods. If either the public key or the private key is set, then the other must also be set for the shared key generation to work.

Notes

GeneratePublicAndPrivateKeys (see page 345) destroys the public and private keys if they are set.

2.7.1.6.4.9.2 - CDiffieHellman::SetParameters Method

Sets the parameters used for generating keys.

C++

```
mxt_result SetParameters(IN const uint8_t* puPrime, IN unsigned int uPrimeSize, IN unsigned int uGenerator, IN
const uint8_t* puPublicKey, IN unsigned int uPublicKeySize, IN const uint8_t* puPrivateKey, IN unsigned int
uPrivateKeySize);
```

Parameters

Parameters	Description
IN const uint8_t* puPrime	The prime modulus.
IN unsigned int uPrimeSize	The size of the prime in bytes.
IN unsigned int uGenerator	The generator. It must be greater or equal to 2.
IN const uint8_t* puPublicKey	The public key.
IN unsigned int uPublicKeySize	The size of the public key in bytes.
IN const uint8_t* puPrivateKey	The private key.
IN unsigned int uPrivateKeySize	The size of the private key in bytes.

Returns

- resS_OK: the parameters has successfully been set.
- resFE_FAIL: failed to set the parameters.
- resFE_INVALID_ARGUMENT: there is a missing mandatory parameter (depending on the context), or their values are invalid.

Description

Sets the prime, the size of the prime in bytes, the generator, the public key, the size of the public key, the private key, and the size of the private key for use in the GeneratePublicAndPrivateKeys (see page 345) and GenerateSharedKey (see page 345) methods. If the public key or the private key parameters are set, then the other parameters must also be set for the shared key generation to work.

Notes

GeneratePublicAndPrivateKeys (see page 345) destroys the public and private keys if they are set.

2.7.1.6.5 - Operators

2.7.1.6.5.1 - CDiffieHellman::= Operator

Copies the referenced class into this one.

C++

```
CDiffieHellman& operator =(IN const CDiffieHellman& rDiffieHellman);
```

Parameters

Parameters	Description
IN const CDiffieHellman& rDiffieHellman	Reference to a CDiffieHellman (see page 340) class.

Returns

Copied CDiffieHellman (see page 340) class.

Description

Copies the reference to this.

2.7.1.7 - CHash Class

Base class for all hash algorithms.

Class Hierarchy



C++

```
class CHash;
```

Description

CHash is the base class for all hash algorithms.

```
void PerformHash(IN CHash* pHash,
```

```

        IN const uint8_t* puInData,
        IN unsigned int uInDataSize,
        OUT uint8_t* puHash)

{
    pHASH->Begin();
    pHASH->Update(puInData, uInDataSize);
    pHASH->End(puHash);
}

```

Location

Crypto/CHash.h

Constructors

Constructor	Description
 CHash (see page 352)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
  ~CHash (see page 352)	Destructor

Legend

	Method
	virtual

Methods

Method	Description
  Begin (see page 353)	Begins a new hash.
  End (see page 353)	Ends the hash.
  GetAlgorithm (see page 354)	Retrieves the algorithm implemented within the inheriting class.
  GetSizeInBits (see page 354)	Retrieves the size in bits of the hash returned by End (see page 353).
  GetSizeInBytes (see page 354)	Retrieves the size in bytes of the hash returned by End (see page 353).
  SetState (see page 354)	Sets the internal state from another instance.
  Update (see page 355)	Updates the hash with new data.

Legend

	Method
	abstract

2.7.1.7.1 - Constructors**2.7.1.7.1.1 - CHash::CHash Constructor**

Constructor.

C++

CHash();

Description

Default constructor.

2.7.1.7.2 - Destructors**2.7.1.7.2.1 - CHash::~CHash Destructor**

Destructor

C++

virtual ~CHash();

Description

Destructor.

2.7.1.7.3 - Methods**2.7.1.7.3.1 - CHash::Begin Method**

Begins a new hash.

C++

```
virtual mxt_result Begin(IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT) = 0;
```

Parameters

Parameters	Description
IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT	Algorithm to use in the hash computation.

Returns

- resS_OK
- resFE_FAIL

Description

Begins a new hash. This is the first method to call to produce a hash. It must be paired with a single call to End (see page 353).

2.7.1.7.3.2 - End**2.7.1.7.3.2.1 - CHash::End Method**

Ends the hash.

C++

```
virtual mxt_result End(OUT CBlob* pHASH) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pHASH	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin (see page 353).

2.7.1.7.3.2.2 - CHash::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT uint8_t* puHash) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL

- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin (see page 353).

2.7.1.7.3.3 - CHash::GetAlgorithm Method

Retrieves the algorithm implemented within the inheriting class.

C++

```
virtual EAlgorithm GetAlgorithm() = 0;
```

Returns

One of the CHash::EAlgorithm (see page 411) values.

Description

Retrieves the algorithm implemented within the inheriting class.

2.7.1.7.3.4 - CHash::GetSizeInBits Method

Retrieves the size in bits of the hash returned by End (see page 353).

C++

```
virtual unsigned int GetSizeInBits() = 0;
```

Returns

The size in bits.

Description

Retrieves the size in bits of the hash returned by End (see page 353).

2.7.1.7.3.5 - CHash::GetSizeInBytes Method

Retrieves the size in bytes of the hash returned by End (see page 353).

C++

```
virtual unsigned int GetSizeInBytes() = 0;
```

Returns

The size in bytes.

Description

Retrieves the size in bytes of the hash returned by End (see page 353).

2.7.1.7.3.6 - CHash::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHash* pHash) = 0;
```

Parameters

Parameters	Description
IN const CHash* pHash	The other hash object instance from which to read the state. The hash algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.7.3.7 - Update

2.7.1.7.3.7.1 - CHash::Update Method

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const CBlob* pData) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Updates the hash with new data. Begin (see page 353) must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.7.3.7.2 - CHash::Update Method

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Input data to hash.
	Size of the input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Updates the hash with new data. Begin (see page 353) must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.8 - CHMac Class

Base class for all HMAC algorithms.

Class Hierarchy

```
CHMac
```

C++

```
class CHMac;
```

Description

CHMac is the base class for all HMAC algorithms.

```
mxt_result PerformHMac(IN const CBlob* pKey,
                        IN const CBlob* pInData,
                        OUT CBlob* pOutHMac,
                        IN CHMac* pHMac)
{
```

```

mxt_result res = ress_OK;

res = pHMac->Begin(pKey);
if (MX_RIS_S(res))
{
    res = pHMac->Update(pInData);
}
if (MX_RIS_S(res))
{
    res = pHMac->End(pOutHMac);
}
}

```

Location

Crypto/CHMac.h

Destructors

Destructor	Description
~CHMac (see page 356)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
  Begin (see page 356)	Begins a new hash message authentication code.
  End (see page 357)	Ends the HMAC generation.
  GetAlgorithm (see page 358)	Gets the algorithm.
  GetSizeInBits (see page 358)	Gets the size in bits.
  GetSizeInBytes (see page 358)	Gets the size in bytes.
  SetState (see page 359)	Sets the internal state from another instance.
  Update (see page 359)	Sets data to HMAC.

Legend

	Method
	abstract

2.7.1.8.1 - Destructors

2.7.1.8.1.1 - CHMac::~CHMac Destructor

Destructor.

C++

```
virtual ~CHMac();
```

Description

Destructor.

2.7.1.8.2 - Methods

2.7.1.8.2.1 - Begin

2.7.1.8.2.1.1 - CHMac::Begin Method

Begins a new hash message authentication code.

C++

```
virtual mxt_result Begin(IN const CBlob* pKey, IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pKey	Pointer to a blob.
IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT	The hashing algorithm to use.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Begins a new hash message authentication code. This is the first method to call to produce a HMAC. It must be paired with a single call to End (see page 357).

See Also

End (see page 357)

2.7.1.8.2.1.2 - CHMac::Begin Method

Begins a new hash message authentication code.

C++

```
virtual mxt_result Begin(IN const uint8_t* puKey, IN unsigned int uKeySize, IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puKey	Key to use to generate the HMAC.
IN unsigned int uKeySize	Size of the key in bytes.
IN EAlgorithm eAlgorithm = eALGORITHM_DEFAULT	The hashing algorithm to use.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Begins a new hash message authentication code. This is the first method to call to produce a HMAC. It must be paired with a single call to End (see page 357).

See Also

End (see page 357)

2.7.1.8.2.2 - End

2.7.1.8.2.2.1 - CHMac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT CBlob* pHash) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pHash	Pointer to a blob to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL

- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.8.2.2 - CHMac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT uint8_t* puHash) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puHash	Pointer to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.8.2.3 - CHMac::GetAlgorithm Method

Gets the algorithm.

C++

```
virtual EAlgorithm GetAlgorithm() = 0;
```

Returns

The algorithm used.

Description

Gets the algorithm used to create the hash.

2.7.1.8.2.4 - CHMac::GetSizeInBits Method

Gets the size in bits.

C++

```
virtual unsigned int GetSizeInBits() = 0;
```

Returns

The size of the HMAC in bits

Description

Gets the size of the generated HMAC in bits.

2.7.1.8.2.5 - CHMac::GetSizeInBytes Method

Gets the size in bytes.

C++

```
virtual unsigned int GetSizeInBytes() = 0;
```

Returns

The size of the HMAC in bytes.

Description

Gets the size of the generated HMAC in bytes.

2.7.1.8.2.6 - CHMac::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHMac* pHmac) = 0;
```

Parameters

Parameters	Description
pHash	The other hash-mac objet instance from which to read the state. The hash-mac algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.8.2.7 - Update**2.7.1.8.2.7.1 - CHMac::Update Method**

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const CBlob* pData) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Pointer to a blob containing the data.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End (see page 357)

2.7.1.8.2.7.2 - CHMac::Update Method

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Data to use to generate the HMAC.
IN unsigned int uDataSize	Size of the data in bytes.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

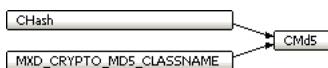
See Also

End (see page 357)

2.7.1.9 - CMd5 Class

Class that manages the MD5 algorithm.

Class Hierarchy



C++

```
class CMd5 : public CHash, public MXD_CRYPTO_MD5_CLASSNAME;
```

Description

CMd5 is the class that manages the MD5 algorithm.

Location

Crypto/CMd5.h

Constructors

Constructor	Description
• CMd5 (see page 361)	Constructor.

CHash Class

CHash Class	Description
• CHash (see page 352)	Constructor.

Legend



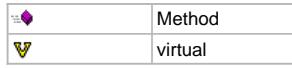
Destructors

Destructor	Description
• ~CMd5 (see page 361)	Destructor.

CHash Class

CHash Class	Description
• ~CHash (see page 352)	Destructor

Legend



Methods

Method	Description
• Begin (see page 361)	See CHash::Begin (see page 353).

• End (see page 362)	Ends the hash.
• GetAlgorithm (see page 362)	Retrieves the algorithm implemented within the inheriting class.
• GetSizeInBits (see page 362)	Retrieves the size in bits of the hash returned by End.
• GetSizeInBytes (see page 363)	Retrieves the size in bytes of the hash returned by End.
• SetState (see page 363)	Sets the internal state from another instance.
• Update (see page 363)	Updates the hash with new data.

CHash Class

CHash Class	Description
• Begin (see page 353)	Begins a new hash.
• End (see page 353)	Ends the hash.
• GetAlgorithm (see page 354)	Retrieves the algorithm implemented within the inheriting class.
• GetSizeInBits (see page 354)	Retrieves the size in bits of the hash returned by End (see page 353).
• GetSizeInBytes (see page 354)	Retrieves the size in bytes of the hash returned by End (see page 353).
• SetState (see page 354)	Sets the internal state from another instance.
• Update (see page 355)	Updates the hash with new data.

Legend

	Method
	virtual
	abstract

2.7.1.9.1 - Constructors

2.7.1.9.1.1 - CMd5::CMd5 Constructor

Constructor.

C++

```
CMd5();
```

Description

Constructor.

2.7.1.9.2 - Destructors

2.7.1.9.2.1 - CMd5::~CMd5 Destructor

Destructor.

C++

```
virtual ~CMd5();
```

Description

Destructor.

2.7.1.9.3 - Methods

2.7.1.9.3.1 - CMd5::Begin Method

See CHash::Begin (see page 353).

C++

```
virtual mxt_result Begin(IN CHash::EAlgorithm eAlgorithm = CHash::eALGORITHM_DEFAULT);
```

2.7.1.9.3.2 - End

2.7.1.9.3.2.1 - CMd5::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.9.3.2.2 - CMd5::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.9.3.3 - CMd5::GetAlgorithm Method

Retrieves the algorithm implemented within the inheriting class.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

One of the CHash::EAlgorithm values.

Description

Retrieves the algorithm implemented within the inheriting class.

2.7.1.9.3.4 - CMd5::GetSizeInBits Method

Retrieves the size in bits of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size in bits.

Description

Retrieves the size in bits of the hash returned by End.

2.7.1.9.3.5 - CMd5::GetSizeInBytes Method

Retrieves the size in bytes of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size in bytes.

Description

Retrieves the size in bytes of the hash returned by End.

2.7.1.9.3.6 - CMd5::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHash* pHash);
```

Parameters

Parameters	Description
pHash	The other hash object instance from which to read the state. The hash algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.9.3.7 - Update**2.7.1.9.3.7.1 - CMd5::Update Method**

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Updates the hash with new data. Begin must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.9.3.7.2 - CMd5::Update Method

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize);
```

Parameters

Parameters	Description
puData	Input data to hash.
uDataSize	Size of the input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

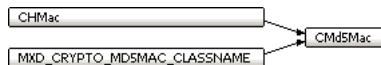
Description

Updates the hash with new data. Begin must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.10 - CMd5Mac Class

Class that manages the MD5 message authentication code algorithm.

Class Hierarchy



C++

```
class CMd5Mac : public CHMac, public MXD_CRYPTO_MD5MAC_CLASSNAME;
```

Description

CMd5Mac is the class that manages the MD5 message authentication code algorithm.

Location

Crypto/CMd5Mac.h

Constructors

Constructor	Description
CMd5Mac (See page 365)	Basic constructor.

Legend



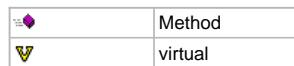
Destructors

Destructor	Description
~CMd5Mac (See page 365)	Destructor.

CHMac Class

CHMac Class	Description
~CHMac (See page 356)	Destructor.

Legend



Methods

Method	Description
Begin (See page 365)	See CHMac::Begin (See page 356).

• End (see page 366)	Ends the HMAC generation.
• GetAlgorithm (see page 366)	Gets the algorithm.
• GetSizeInBits (see page 367)	Gets the size in bits.
• GetSizeInBytes (see page 367)	Gets the size in bytes.
• SetState (see page 367)	Sets the internal state from another instance.
• Update (see page 367)	Sets data to HMAC.

CHMac Class

CHMac Class	Description
• Begin (see page 356)	Begins a new hash message authentication code.
• End (see page 357)	Ends the HMAC generation.
• GetAlgorithm (see page 358)	Gets the algorithm.
• GetSizeInBits (see page 358)	Gets the size in bits.
• GetSizeInBytes (see page 358)	Gets the size in bytes.
• SetState (see page 359)	Sets the internal state from another instance.
• Update (see page 359)	Sets data to HMAC.

Legend

	Method
	virtual
	abstract

2.7.1.10.1 - Constructors

2.7.1.10.1.1 - CMd5Mac::CMd5Mac Constructor

Basic constructor.

C++

```
CMd5Mac();
```

Description

Basic constructor.

2.7.1.10.2 - Destructors

2.7.1.10.2.1 - CMd5Mac::~CMd5Mac Destructor

Destructor.

C++

```
virtual ~CMd5Mac();
```

Description

Destructor.

2.7.1.10.3 - Methods

2.7.1.10.3.1 - Begin

2.7.1.10.3.1.1 - CMd5Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const CBlob* pKey, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.10.3.1.2 - CMd5Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const uint8_t* puKey, IN unsigned int uKeySize, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.10.3.2 - End

2.7.1.10.3.2.1 - CMd5Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	Pointer to a blob to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.10.3.2.2 - CMd5Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	Pointer to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.10.3.3 - CMd5Mac::GetAlgorithm Method

Gets the algorithm.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

The algorithm used.

Description

Gets the algorithm used to create the hash.

2.7.1.10.3.4 - CMd5Mac::GetSizeInBits Method

Gets the size in bits.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size of the HMAC in bits

Description

Gets the size of the generated HMAC in bits.

2.7.1.10.3.5 - CMd5Mac::GetSizeInBytes Method

Gets the size in bytes.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size of the HMAC in bytes.

Description

Gets the size of the generated HMAC in bytes.

2.7.1.10.3.6 - CMd5Mac::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHMac* pHMac);
```

Parameters

Parameters	Description
pHash	The other hash-mac objet instance from which to read the state. The hash-mac algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.10.3.7 - Update**2.7.1.10.3.7.1 - CMd5Mac::Update Method**

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Pointer to a blob containing the data.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End

2.7.1.10.3.7.2 - CMd5Mac::Update Method

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize);
```

Parameters

Parameters	Description
puData	Data to use to generate the HMAC.
uDataSize	Size of the data in bytes.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End

2.7.1.11 - CPrivateKey Class

Class that manages the private key for all algorithms.

Class Hierarchy



C++

```
class CPrivateKey : public MXD_CRYPTO_PRIVATEKEY_CLASSNAME;
```

Description

CPrivateKey is the class that manages the private key for all algorithms. The private key is mostly used for authentication purposes.

Location

Crypto/CPrivateKey.h

See Also

CPublicKey (see page 374)

Constructors

Constructor	Description
• CPrivateKey (see page 369)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CPrivateKey (see page 370)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 373)	Different than algorithm.
 = (see page 374)	Assignment operator.
 == (see page 374)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
 GetAlgorithm (see page 370)	Returns the crypto algorithm associated with the key.
 Restore (see page 370)	Restores a key from its serialized state.
 RestoreDer (see page 371)	Restores a private key for its serialized state DER format.
 RestorePem (see page 372)	Restores a private key for its serialized state PEM format.
 Store (see page 372)	Stores a key to its serialized state.
 StoreDer (see page 373)	Stores a private key to its serialized state DER format.
 StorePem (see page 373)	Stores a private key to its serialized state PEM format.

Legend

	Method
--	--------

2.7.1.11.1 - Constructors**2.7.1.11.1.1 - CPrivateKey****2.7.1.11.1.1.1 - CPrivateKey::CPrivateKey Constructor**

Constructor.

C++

```
CPrivateKey();
```

Description

Default constructor.

2.7.1.11.1.1.2 - CPrivateKey::CPrivateKey Constructor

Copy constructor.

C++

```
CPrivateKey(IN const CPrivateKey& rKey);
```

Parameters

Parameters	Description
IN const CPrivateKey& rKey	Reference to the key from which to copy.

Description

Copy constructor.

2.7.1.11.1.1.3 - CPrivateKey::CPrivateKey Constructor

Copy constructor.

C++

```
CPrivateKey(IN const CPrivateKey* pKey);
```

Parameters

Parameters	Description
IN const CPrivateKey* pKey	Pointer to the key from which to copy.

Description

Copy constructor.

2.7.1.11.2 - Destructors**2.7.1.11.2.1 - CPrivateKey::~CPrivateKey Destructor**

Destructor.

C++

```
virtual ~CPrivateKey();
```

Description

Destructor.

2.7.1.11.3 - Methods**2.7.1.11.3.1 - CPrivateKey::GetAlgorithm Method**

Returns the crypto algorithm associated with the key.

C++

```
mxt_result GetAlgorithm(OUT EAlgorithm* peAlgorithm) const;
```

Parameters

Parameters	Description
OUT EAlgorithm* peAlgorithm	Pointer to contain the algorithm.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the algorithm for which this key is used.

2.7.1.11.3.2 - Restore**2.7.1.11.3.2.1 - CPrivateKey::Restore Method**

Restores a key from its serialized state.

C++

```
mxt_result Restore(IN const CBlob* pblobKey, IN EAlgorithm eAlgorithm, IN IPassPhrase* pPassPhrase, IN
```

```
mxt_opaque opqPassPhraseParameter);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN EAlgorithm eAlgorithm	The algorithm for which the key is used.
IN IPassPhrase* pPassPhrase	Passphrase interface used to parse the object.
IN mxt_opaque opqPassPhraseParameter	Parameter to pass to the passphrase interface.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.11.3.2.2 - CPrivateKey::Restore Method

Restores a key from its serialized state.

C++

```
mxt_result Restore(IN const CBlob* pblobKey, IN EAlgorithm eAlgorithm, IN const char* pszPassPhrase);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN EAlgorithm eAlgorithm	The algorithm for which the key is used.
IN const char* pszPassPhrase	Pointer to the passphrase to restore the key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.11.3.3 - CPrivateKey::RestoreDer Method

Restores a private key for its serialized state DER format.

C++

```
mxt_result RestoreDer(IN const CBlob* pblobKey, IN EAlgorithm eAlgorithm);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN EAlgorithm eAlgorithm	The algorithm for which the key is used.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.11.3.4 - RestorePem

2.7.1.11.3.4.1 - CPrivateKey::RestorePem Method

Restores a private key for its serialized state PEM format.

C++

```
mxt_result RestorePem(IN const CBlob* pblobKey, IN IPassPhrase* pPassPhrase, IN mxt_opaque opqPassPhraseParameter);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN IPassPhrase* pPassPhrase	Passphrase interface used to parse the object.
IN mxt_opaque opqPassPhraseParameter	Parameter to pass to the passphrase interface.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.11.3.4.2 - CPrivateKey::RestorePem Method

Restores a private key for its serialized state PEM format.

C++

```
mxt_result RestorePem(IN const CBlob* pblobKey, IN const char* pszPassPhrase);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN const char* pszPassPhrase	Passphrase used to serialize.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.11.3.5 - CPrivateKey::Store Method

Stores a key to its serialized state.

C++

```
mxt_result Store(OUT CBlob* pblobKey, IN EEncoding eEncoding, IN const char* pszPassPhrase) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.
IN EEncoding eEncoding	Encoding type to use to serialize the key.
IN const char* pszPassPhrase	Pointer to the passphrase to use to serialize the object.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.11.3.6 - CPrivateKey::StoreDer Method

Stores a private key to its serialized state DER format.

C++

```
mxt_result StoreDer(OUT CBlob* pblobKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.11.3.7 - CPrivateKey::StorePem Method

Stores a private key to its serialized state PEM format.

C++

```
mxt_result StorePem(OUT CBlob* pblobKey, IN const char* pszPassPhrase) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.
IN const char* pszPassPhrase	Passphrase used to serialize.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.11.4 - Operators**2.7.1.11.4.1 - CPrivateKey::!= Operator**

Different than algorithm.

C++

```
bool operator !=(IN const CPrivateKey& rKey) const;
```

Parameters

Parameters	Description
IN const CPrivateKey& rKey	Reference to the key to copy.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.7.1.11.4.2 - CPrivateKey::= Operator

Assignment operator.

C++

```
CPrivateKey& operator =(IN const CPrivateKey& rKey);
```

Parameters

Parameters	Description
IN const CPrivateKey& rKey	Reference to the key to assign.

Returns

A reference to the assigned key.

Description

Assigns the right hand key to the left hand one.

2.7.1.11.4.3 - CPrivateKey::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CPrivateKey& rKey) const;
```

Parameters

Parameters	Description
IN const CPrivateKey& rKey	Reference to the key with which to compare.

Returns

True if both objects are the same, false otherwise.

Description

Verifies that both objects are equal.

2.7.1.12 - CPublicKey Class

Class that manages the public key for all algorithms.

Class Hierarchy**C++**

```
class CPublicKey : public MXD_CRYPTO_PUBLICKEY_CLASSNAME;
```

Description

CPublicKey is the class that manages the public key for all algorithms. The public key is mostly used for authentication purposes.

Location

Crypto/CPublicKey.h

See Also

CPrivateKey (see page 368)

Constructors

Constructor	Description
 CPublicKey (see page 375)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CPublicKey (see page 376)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 379)	Different than algorithm.
 = (see page 379)	Assignment operator.
 == (see page 379)	Comparison operator.

Legend

	Method
---	--------

Methods

Method	Description
 GetAlgorithm (see page 376)	Returns the crypto algorithm associated with the key.
 Restore (see page 376)	Restores a key from its serialized state.
 RestoreDer (see page 377)	Restores a public key for its serialized state DER format.
 RestorePem (see page 377)	Restores a public key for its serialized state PEM format.
 Store (see page 377)	Stores a key to its serialized state.
 StoreDer (see page 378)	Stores a public key to its serialized state DER format.
 StorePem (see page 378)	Stores a public key to its serialized state PEM format.

Legend

	Method
---	--------

2.7.1.12.1 - Constructors

2.7.1.12.1.1 - CPublicKey

2.7.1.12.1.1.1 - CPublicKey::CPublicKey Constructor

Constructor.

C++

```
CPublicKey();
```

Description

Default constructor.

2.7.1.12.1.1.2 - CPublicKey::CPublicKey Constructor

Copy constructor.

C++

```
CPublicKey(IN const CPublicKey& rKey);
```

Parameters

Parameters	Description
IN const CPublicKey& rKey	Reference to the key from which to copy.

Description

Copy constructor.

2.7.1.12.1.1.3 - CPublicKey::CPublicKey Constructor

Copy constructor.

C++

```
CPublicKey(IN const CPublicKey* pKey);
```

Parameters

Parameters	Description
IN const CPublicKey* pKey	Pointer to the key from which to copy.

Description

Copy constructor.

2.7.1.12.2 - Destructors**2.7.1.12.2.1 - CPublicKey::~CPublicKey Destructor**

Destructor.

C++

```
virtual ~CPublicKey();
```

Description

Destructor.

2.7.1.12.3 - Methods**2.7.1.12.3.1 - CPublicKey::GetAlgorithm Method**

Returns the crypto algorithm associated with the key.

C++

```
mxt_result GetAlgorithm(OUT EAlgorithm* peAlgorithm) const;
```

Parameters

Parameters	Description
OUT EAlgorithm* peAlgorithm	Pointer to contain the algorithm.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the algorithm for which this key is used.

2.7.1.12.3.2 - CPublicKey::Restore Method

Restores a key from its serialized state.

C++

```
mxt_result Restore(IN const CBlob* pblobKey, IN EAlgorithm eAlgorithm);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN EAlgorithm eAlgorithm	The algorithm for which the key is used.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.12.3.3 - CPublicKey::RestoreDer Method

Restores a public key for its serialized state DER format.

C++

```
mxt_result RestoreDer(IN const CBlob* pblobKey, IN EAlgorithm eAlgorithm);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.
IN EAlgorithm eAlgorithm	The algorithm for which the key is used.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.12.3.4 - CPublicKey::RestorePem Method

Restores a public key for its serialized state PEM format.

C++

```
mxt_result RestorePem(IN const CBlob* pblobKey);
```

Parameters

Parameters	Description
IN const CBlob* pblobKey	Pointer to the blob containing the key to restore.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a key from its serialized state.

2.7.1.12.3.5 - CPublicKey::Store Method

Stores a key to its serialized state.

C++

```
mxt_result Store(OUT CBlob* pblobKey, IN EEncoding eEncoding) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.
IN EEncoding eEncoding	Encoding type to use to serialize the key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.12.3.6 - CPublicKey::StoreDer Method

Stores a public key to its serialized state DER format.

C++

```
mxt_result StoreDer(OUT CBlob* pblobKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.12.3.7 - CPublicKey::StorePem Method

Stores a public key to its serialized state PEM format.

C++

```
mxt_result StorePem(OUT CBlob* pblobKey) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobKey	Pointer to the blob to contain the key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores a key in its serialized state.

2.7.1.12.4 - Operators

2.7.1.12.4.1 - CPublicKey::!= Operator

Different than algorithm.

C++

```
bool operator !=(IN const CPublicKey& rKey) const;
```

Parameters

Parameters	Description
IN const CPublicKey& rKey	Reference to the key to copy.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.7.1.12.4.2 - CPublicKey::= Operator

Assignment operator.

C++

```
CPublicKey& operator =(IN const CPublicKey& rKey);
```

Parameters

Parameters	Description
IN const CPublicKey& rKey	Reference to the key to assign.

Returns

A reference to the assigned key.

Description

Assigns the right hand key to the left hand one.

2.7.1.12.4.3 - CPublicKey::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CPublicKey& rKey) const;
```

Parameters

Parameters	Description
IN const CPublicKey& rKey	Reference to the key with which to compare.

Returns

True if both objects are the same, false otherwise.

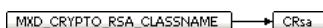
Description

Verifies that both objects are equal.

2.7.1.13 - CRsa Class

Class that manages the RSA algorithm.

Class Hierarchy



C++

```
class CRsa : public MXD_CRYPTO_RSA_CLASSNAME;
```

Description

CRsa is the class that manages RSA encryption, decryption, signature, signature verification, and key management.

RSA: RSA is a public key algorithm used for data encryption and digital signatures.

The encryption of a message with RSA is a very costly process and is many orders of magnitude slower than symmetric encryption. For this reason, RSA is used for sharing the keys used in symmetric algorithms as both users must be in possession of a common key to use these algorithms.

It is also used to digitally sign a message as anybody can authenticate that a user has signed a document using his private key.

A RSA private/public key pair is generated in the following way: A user chooses an exponent e to be used. This exponent can be anything but some are better suited as they speed up calculations. Two random large primes p and q are then chosen and multiplied to give the modulus n. e and n both form the public key. e and $(p - 1)(q - 1)$ must be relatively prime.

The private key d is then calculated: $d = e^{-1} (p - 1)(q - 1)$.

The two primes p and q should be discarded at this point as they are of no more use once the public/private key pair has been generated.

Key sharing: Bob encrypts a key to be used in a symmetric cryptographic algorithm with Alice using Alice's public key and sends it to Alice. Only Alice can decrypt this message since she is the only one with her private key. Once Alice has the shared key, she can communicate with Bob using the new key for the symmetric algorithm. Bob obtains Alice's public key either from a key broker or Alice's certificate.

Initiator	Acceptor
---Get acceptor public key ---	
<pre>PublicKeyEncrypt(PublicKey, MessageToEncrypt, PaddingType, EncryptedMessage)</pre>	
<pre>PrivateKeyDecrypt(EncryptedMessage, PaddingType, DecryptedMessage)</pre>	

Signature: Using her private key, Alice encrypts a hashed message. She can then send the signed hash and the hashed message to Bob, who can verify that this message is really from Alice by decompressing the encrypted message using Alice's public key and comparing it to the hashed message. If both of these are the same, then the message hash can only have been created by Alice. This uniquely identifies Alice as the creator.

Initiator	Acceptor
Sign(MessageHash,	
DigestType,	
SignedMessage)	
<pre>Verify(PublicKey, EncryptedMessage, DigestType, MessageHash)</pre>	

Location

Crypto/CRsa.h

Constructors

Constructor	Description
CRsa (see page 381)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CRsa (see page 381)	Default destructor.

Legend

	Method
	virtual

Methods

Method	Description
GenerateKey (See page 381)	Generates the private key and the product of two primes.
GetPrivateKey (See page 382)	Gets the private key.
GetPublicKey (See page 382)	Gets the public key.
PrivateKeyDecrypt (See page 382)	Decrypts the message with the private key.
PrivateKeyEncrypt (See page 383)	Encrypts the message with the private key.
PublicKeyDecrypt (See page 384)	Decrypts the message with the public key.
PublicKeyEncrypt (See page 386)	Encrypts the message with the public key.
SetKey (See page 387)	Sets the keys for RSA.
Sign (See page 387)	Signs the hash of a message.
Verify (See page 388)	Verifies a signature.

Legend

	Method
--	--------

2.7.1.13.1 - Constructors

2.7.1.13.1.1 - CRsa::CRsa Constructor

Constructor.

C++

```
CRsa();
```

Description

Default constructor.

2.7.1.13.2 - Destructors

2.7.1.13.2.1 - CRsa::~CRsa Destructor

Default destructor.

C++

```
virtual ~CRsa();
```

Description

Default destructor.

2.7.1.13.3 - Methods

2.7.1.13.3.1 - CRsa::GenerateKey Method

Generates the private key and the product of two primes.

C++

```
mxt_result GenerateKey(IN unsigned int uExponent, IN unsigned int uKeySizeInBits);
```

Parameters

Parameters	Description
IN unsigned int uExponent	The exponent of the public key.
IN unsigned int uKeySizeInBits	The size of the key to generate in number of bits.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method generates 2 large prime numbers randomly, p and q. Then, p and q are used to compute n, the product of 2 primes also

referred to as the modulus. The exponent, e, supplied in uExponent together with the modulus, form the public key.

The private key, d, is then calculated using the following formula: $d = e^{-1} \bmod(p - 1)(q - 1)$.

Notes: The public exponent must be an odd large number as this greatly increases the chance that e and d are relatively prime. There are a few common exponents that are recommended by PKCS#1, PEM, and x.509. These are: 3, 17, and 65537. These numbers speed up the exponentiation operations.

The recommendation of these exponents explains why uExponent is defined as an unsigned integer even if it is specified the exponent may be a large number.

2.7.1.13.3.2 - CRsa::GetPrivateKey Method

Gets the private key.

C++

```
mxt_result GetPrivateKey(OUT CPrivateKey* pPrivateKey);
```

Parameters

Parameters	Description
OUT CPrivateKey* pPrivateKey	Pointer to a CPrivateKey (see page 368) class to receive the private key.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method is used to get the private key currently associated with this CRsa (see page 379) instance. The key is returned in the form of a CPrivateKey (see page 368) class for abstraction.

2.7.1.13.3.3 - CRsa::GetPublicKey Method

Gets the public key.

C++

```
mxt_result GetPublicKey(OUT CPublicKey* pPublicKey);
```

Parameters

Parameters	Description
OUT CPublicKey* pPublicKey	Pointer to a CPublicKey (see page 374) to contain the exponent.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method is used to get the public key currently associated with this CRsa (see page 379) instance. The key is returned in the form of a CPublicKey (see page 374) class for abstraction.

Notes: The key returned in CPublicKey (see page 374) is valid as long as the m_pRsa is also valid. Changing the public key either in the CPublicKey (see page 374) or in the CRsa (see page 379) results in an unknown behaviour of both the CPublicKey (see page 374) and the CRsa (see page 379).

2.7.1.13.3.4 - PrivateKeyDecrypt

2.7.1.13.3.4.1 - CRsa::PrivateKeyDecrypt Method

Decrypts the message with the private key.

C++

```
mxt_result PrivateKeyDecrypt(IN const CBlob* pblobEncryptedMessage, IN EEncryptionPaddingAlgorithm ePadding, OUT CBlob* pblobMessage);
```

Parameters

Parameters	Description
IN const CBlob* pblobEncryptedMessage	Pointer to a blob containing the message to decrypt.

IN EEncryptionPaddingAlgorithm ePadding	The type of padding used on the message.
OUT CBlob* pblobMessage	Pointer to a blob to contain the decrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method decrypts the message referred by pblobEncryptedMessage using the private key stored inside this CRsa (see page 379) instance.

In the end, the decrypted message and its size are stored inside a CBlob (see page 95) at the location pointed by pblobMessage. If it is required, its capacity is increased to provide enough space to store the decrypted message.

2.7.1.13.3.4.2 - CRsa::PrivateKeyDecrypt Method

Decrypts the message with the private key.

C++

```
mxt_result PrivateKeyDecrypt(IN const uint8_t* puEncryptedMessage, IN unsigned int uEncryptedMessageSize, IN EEncryptionPaddingAlgorithm ePadding, IN unsigned int uMessageCapacity, OUT uint8_t* puMessage, OUT unsigned int* puMessageSize);
```

Parameters

Parameters	Description
IN const uint8_t* puEncryptedMessage	Pointer to a byte array containing the message to decrypt.
IN unsigned int uEncryptedMessageSize	The size of the message to decrypt.
IN EEncryptionPaddingAlgorithm ePadding	The type of padding used on the message.
IN unsigned int uMessageCapacity	The capacity of the array to receive decrypted message.
OUT uint8_t* puMessage	Pointer to a byte array to contain the decrypted message.
OUT unsigned int* puMessageSize	The size of the decrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method decrypts the message referred by puEncryptedMessage using the private key stored in this CRsa (see page 379) instance.

In the end, the decrypted message is stored at the location pointed by puMessage. The integer pointed by puMessageSize is updated to reflect the size of the decrypted message. The size does not exceed the capacity indicated by uMessageCapacity. An error is returned if the capacity is too small.

If puMessage is NULL, the decryption process does not occur. Instead, the value pointed by puMessageSize is updated to reflect the maximum size required to store the decrypted message. This gives the caller the opportunity to properly allocate the buffer.

2.7.1.13.3.5 - PrivateKeyEncrypt

2.7.1.13.3.5.1 - CRsa::PrivateKeyEncrypt Method

Encrypts the message with the private key.

C++

```
mxt_result PrivateKeyEncrypt(IN const CBlob* pblobMessage, IN ESignaturePaddingAlgorithm ePadding, OUT CBlob* pblobEncryptedMessage);
```

Parameters

Parameters	Description
IN const CBlob* pblobMessage	Pointer to a blob containing the message to encrypt.
IN ESignaturePaddingAlgorithm ePadding	The type of padding used on the message.
OUT CBlob* pblobEncryptedMessage	Pointer to a blob to contain the encrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method encrypts the message pointed by pblobMessage using the private key stored in this instance of CRsa (see page 379). The encryption of a message with a private key confirms at the same time the identity of the source to anyone who decrypts its message successfully with the public key.

In the end, the encrypted message and its size are stored inside a CBlob (see page 95) at the location pointed by pblobEncryptedMessage. If it is required, its capacity is increased to provide enough space to store the encrypted message.

The parameter ePadding configures the type of padding used for the encryption with respect to ESignaturePaddingAlgorithm. This padding reduces the space available to store the encrypted message. For each padding type, uMessageSize must be less than or equal to a certain amount: ePKCS1: The message size must be less than or equal to the size of the modulus (private key size) minus 11 padding bytes.

eNOPADDING: The message size must be equal to the size of the modulus (private key size).

This method is also useful for generating signature.

2.7.1.13.3.5.2 - CRsa::PrivateKeyEncrypt Method

Encrypts the message with the private key.

C++

```
mxt_result PrivateKeyEncrypt(IN const uint8_t* puMessage, IN unsigned int uMessageSize, IN
ESignaturePaddingAlgorithm ePadding, IN unsigned int uEncryptedMessageCapacity, OUT uint8_t* puEncryptedMessage,
OUT unsigned int* puEncryptedMessageSize);
```

Parameters

Parameters	Description
IN const uint8_t* puMessage	Pointer to a byte array containing the message to encrypt.
IN unsigned int uMessageSize	The size of the message to encrypt.
IN ESignaturePaddingAlgorithm ePadding	The type of padding used on the message.
IN unsigned int uEncryptedMessageCapacity	The capacity of the buffer supplied to store encrypted message.
OUT uint8_t* puEncryptedMessage	Pointer to a byte array to contain the encrypted message.
OUT unsigned int* puEncryptedMessageSize	The size of the encrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method encrypts the message pointed by puMessage using the private key stored in this instance of CRsa (see page 379). The encryption of a message with a private key confirms at the same time the identity of the source to anyone who decrypts its message successfully with the public key.

In the end, the encrypted message is stored at the location pointed by puEncryptedMessage. The integer pointed by puEncryptedMessageSize is updated to reflect the size of the encrypted message. The size does not exceed the capacity indicated by uEncryptedMessageCapacity. An error is returned if the capacity is too small.

The parameter ePadding configures the type of padding used for the encryption with respect to ESignaturePaddingAlgorithm. This padding reduces the space available to store the encrypted message. For each padding type, uMessageSize must be less than or equal to a certain amount: ePKCS1: The message size must be less than or equal to the size of the modulus (private key size) minus 11 padding bytes.

eNOPADDING: The message size must be equal to the size of the modulus (private key size).

If puEncryptedMessage is NULL, the encryption process does not occur. Instead, the value pointed by puEncryptedMessageSize is updated to reflect the maximum size required to store the encrypted message. This gives the caller the opportunity to properly allocate the buffer.

This method is also useful for signing a message.

2.7.1.13.3.6 - PublicKeyDecrypt

2.7.1.13.3.6.1 - CRsa::PublicKeyDecrypt Method

Decrypts the message with the public key.

C++

```
mxt_result PublicKeyDecrypt(IN const CPublicKey* pRemotePublicKey, IN const CBlob* pblobEncryptedMessage, IN
ESignaturePaddingAlgorithm ePadding, OUT CBlob* pblobMessage);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a CPublicKey (see page 374) class containing the key to be used.
IN const CBlob* pblobEncryptedMessage	Blob to decrypt using the public key.
IN ESignaturePaddingAlgorithm ePadding	The type of padding used on the message.
OUT CBlob* pblobMessage	Blob to contain the decrypted bytes.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method decrypts the message referred by pblobEncryptedMessage using the public key supplied with pRemotePublicKey. The successful decryption of a message with a public key confirms at the same time the authenticity of the source that encrypted the message.

In the end, the decrypted message and its size are stored inside a CBlob (see page 95) at the location pointed by pblobMessage. If it is required, its capacity is increased to provide enough space to store the decrypted message.

This method is also useful for signature verification.

See Also

Verify (see page 388)

2.7.1.13.3.6.2 - CRsa::PublicKeyDecrypt Method

Decrypts the message with the public key.

C++

```
mxt_result PublicKeyDecrypt(IN const CPublicKey* pRemotePublicKey, IN const uint8_t* puEncryptedMessage, IN
unsigned int uEncryptedMessageSize, IN ESignaturePaddingAlgorithm ePadding, IN unsigned int uMessageCapacity,
OUT uint8_t* puMessage, OUT unsigned int* puMessageSize);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a CPublicKey (see page 374) class containing the key to be used.
IN const uint8_t* puEncryptedMessage	Pointer to a byte array to decrypt using the public key.
IN unsigned int uEncryptedMessageSize	The size of the byte array to decrypt.
IN ESignaturePaddingAlgorithm ePadding	The type of padding used on the message.
IN unsigned int uMessageCapacity	The capacity of the buffer supplied to receive the decrypted message.
OUT uint8_t* puMessage	Pointer to an array to contain the decrypted bytes.
OUT unsigned int* puMessageSize	The size of the decrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method decrypts the message referred by puEncryptedMessage using the public key supplied with pRemotePublicKey. The successful decryption of a message with a public key confirms at the same time the authenticity of the source that encrypted the message.

In the end, the decrypted message is stored at the location pointed by puMessage. The integer pointed by puMessageSize is updated to reflect the size of the decrypted message. The size does not exceed the capacity indicated by uMessageCapacity. An error is returned if the capacity is too small.

If puMessage is NULL, the decryption process does not occur. Instead, the value pointed by puMessageSize is updated to reflect the maximum size required to store the decrypted message. This gives the caller the opportunity to properly allocate the buffer.

This method is also useful for signature verification.

See Also

Verify (see page 388)

2.7.1.13.3.7 - PublicKeyEncrypt

2.7.1.13.3.7.1 - CRsa::PublicKeyEncrypt Method

Encrypts the message with the public key.

C++

```
mxt_result PublicKeyEncrypt(IN const CPublicKey* pRemotePublicKey, IN const CBlob* pblobMessage, IN
EEncryptionPaddingAlgorithm ePadding, OUT CBlob* pblobEncryptedMessage);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a CPublicKey (see page 374) class containing the key to be used.
IN const CBlob* pblobMessage	Pointer to a blob to encrypt using the public key.
IN EEncryptionPaddingAlgorithm ePadding	The type of padding used on the message.
OUT CBlob* pblobEncryptedMessage	Pointer to a blob to contain the encrypted message.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method encrypts the message pointed by pblobMessage using the public key pointed by the pRemotePublicKey parameter. In the end, the encrypted message and its size are stored inside a CBlob (see page 95) at the location pointed by pblobEncryptedMessage. If it is required, its capacity is increased to provide enough space to store the encrypted message.

The parameter ePadding configures the type of padding used for the encryption with respect to EEncryptionPaddingAlgorithm. This padding reduces the space available to store the encrypted message. For each padding type, uMessageSize must be less than or equal to a certain amount: ePKCS1: The message size must be less than or equal to the size of the modulus (public key size) minus 11 padding bytes.

ePKCS1_OAEP: The message size must be less than or equal to the size of the modulus (public key size) minus 41 padding bytes.

eNOPADDING: The message size must be equal to the size of the modulus (public key size).

2.7.1.13.3.7.2 - CRsa::PublicKeyEncrypt Method

Encrypts the message with the public key.

C++

```
mxt_result PublicKeyEncrypt(IN const CPublicKey* pRemotePublicKey, IN const uint8_t* puMessage, IN unsigned int
uMessageSize, IN EEncryptionPaddingAlgorithm ePadding, IN unsigned int uEncryptedMessageCapacity, OUT uint8_t*
puEncryptedMessage, OUT unsigned int* puEncryptedMessageSize);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a CPublicKey (see page 374) class containing the key to be used.
IN const uint8_t* puMessage	Pointer to a byte array to encrypt using the public key.
IN unsigned int uMessageSize	The size of the byte array to encrypt.
IN EEncryptionPaddingAlgorithm ePadding	The type of padding used on the message.
IN unsigned int uEncryptedMessageCapacity	The capacity of the buffer supplied to store the encrypted message.
OUT uint8_t* puEncryptedMessage	Pointer to a byte array to contain the encrypted bytes.
OUT unsigned int* puEncryptedMessageSize	The size of the encrypted byte array.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method encrypts the message pointed by puMessage using the public key pointed by the pRemotePublicKey parameter. In the end, the encrypted message is stored at the location pointed by puEncryptedMessage. The integer pointed by puEncryptedMessageSize is updated to reflect the size of the encrypted message. The size does not exceed the capacity indicated by uEncryptedMessageCapacity. An error is returned if the capacity is too small.

If puEncryptedMessage is NULL, the encryption process does not occur. Instead, the value pointed by puEncryptedMessageSize is updated to reflect the maximum size required to store the encrypted message. This gives the caller the opportunity to properly allocate

the buffer.

The parameter ePadding configures the type of padding used for the encryption with respect to EEncryptionPaddingAlgorithm. This padding reduces the space available to store the encrypted message. For each padding type, uMessageSize must be less than or equal to a certain amount: ePKCS1: The message size must be less than or equal to the size of the modulus (public key size) minus 11 padding bytes.

ePKCS1_OAEP: The message size must be less than or equal to the size of the modulus (public key size) minus 41 padding bytes.

eNOPADDING: The message size must be equal to the size of the modulus (public key size).

2.7.1.13.3.8 - CRsa::SetKey Method

Sets the keys for RSA.

C++

```
mxt_result SetKey(IN const CPrivateKey* pPrivateKey);
```

Parameters

Parameters	Description
IN const CPrivateKey* pPrivateKey	Pointer to a CPrivateKey (see page 368).

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method sets the public and private keys for the current instance of CRsa (see page 379). The key information supplied in the parameter pPrivateKey is used to determine the public key and the private key.

When this method is used, the private key was generated by other means and the object is configured to use that key.

2.7.1.13.3.9 - Sign

2.7.1.13.3.9.1 - CRsa::Sign Method

Signs the hash of a message.

C++

```
mxt_result Sign(IN const CBlob* pblobHash, IN CHash::EAlgorithm eHashAlgorithm, OUT CBlob* pblobSignature);
```

Parameters

Parameters	Description
IN const CBlob* pblobHash	Pointer to a blob containing the hash to sign.
OUT CBlob* pblobSignature	Pointer to a blob to contain the signature.
eAlgorithm	Hash algorithm used to generate the hash.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method signs the hash pointed by pblobHash with the private key stored inside this CRsa (see page 379) instance.

In the end, a CBlob (see page 95) pointed by pblobSignature is updated to contain the signature and its corresponding size. If required, the capacity of the object pointed by pblobSignature is increased so there is enough space to hold the signature.

The parameter eHashAlgorithm indicates the hash algorithm used to generate pblobHash with respect to CHash::EAlgorithm (see page 411).

2.7.1.13.3.9.2 - CRsa::Sign Method

Signs the hash of a message.

C++

```
mxt_result Sign(IN const uint8_t* puHash, IN unsigned int uHashSize, IN CHash::EAlgorithm eHashAlgorithm, IN
unsigned int uSignatureCapacity, OUT uint8_t* puSignature, OUT unsigned int* puSignatureSize);
```

Parameters

Parameters	Description
IN const uint8_t* puHash	Pointer to a byte array containing the hash to sign.
IN unsigned int uHashSize	The size of the hash to sign.
IN unsigned int uSignatureCapacity	The capacity of the array to receive signature.
OUT uint8_t* puSignature	Pointer to a byte array to contain the signature.
OUT unsigned int* puSignatureSize	The size of the signature.
eAlgorithm	Hashing algorithm used to generate the hash.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method signs the hash pointed by puHash with the private key stored inside this CRsa (see page 379) instance.

In the end, the hash signature is stored at the location pointed by puSignature. The integer pointed by puSignatureSize is updated to reflect the size of the signature. The size does not exceed the capacity indicated by uSignatureCapacity. An error is returned if the capacity is too small.

If puSignature is NULL, the signature process does not occur. Instead, the value pointed by puSignatureSize is updated to reflect the maximum size required to store the signature. It gives the caller the opportunity to properly allocate the buffer.

The parameter eHashAlgorithm indicates the hash algorithm used to generate puHash with respect to CHash::EAlgorithm (see page 411).

2.7.1.13.3.10 - Verify

2.7.1.13.3.10.1 - CRsa::Verify Method

Verifies a signature.

C++

```
mxt_result Verify(IN const CPublicKey* pRemotePublicKey, IN const CBlob* pblobSignature, IN const CBlob* pblobHash, IN CHash::EAlgorithm eHashAlgorithm);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a CPublicKey (see page 374) containing the remote key.
IN const CBlob* pblobSignature	Pointer to a blob containing the signed message.
IN const CBlob* pblobHash	Pointer to a blob containing the original hash.
eAlgorithm	Hash algorithm used.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

This method decrypts the signatures pointed by pblobSignature with the public key supplied by pRemotePublicKey and verifies that the resulting hash corresponds to the hash referred by pblobHash.

The parameter eHashAlgorithm is also compared against the hash algorithm used to generate the signature.

2.7.1.13.3.10.2 - CRsa::Verify Method

Verifies a signature.

C++

```
mxt_result Verify(IN const CPublicKey* pRemotePublicKey, IN const uint8_t* puSignature, IN unsigned int uSignatureSize, IN const uint8_t* puHash, IN unsigned int uHashSize, IN CHash::EAlgorithm eHashAlgorithm);
```

Parameters

Parameters	Description
IN const CPublicKey* pRemotePublicKey	Pointer to a blob containing the remote public key.
IN const uint8_t* puSignature	Pointer to a byte array containing the signature to verify.

IN unsigned int uSignatureSize	The size of the signature to verify.
IN const uint8_t* puHash	Pointer to a byte array containing the original hash.
IN unsigned int uHashSize	The size of the hash to compare with the signature.
eAlgorithm	Hash algorithm used.

Returns

resS_OK if successful, resFE_FAIL or resFE_INVALID_ARGUMENT otherwise.

Description

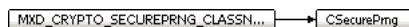
This method decrypts the signatures pointed by puSignature with the public key supplied by pRemotePublicKey and verifies that the resulting hash corresponds to the hash referred by puHash.

The parameter eHashAlgorithm is also compared against the hash algorithm used to generate the signature.

2.7.1.14 - CSecurePrng Class

Class implementing a secure pseudo-random generator.

Class Hierarchy



C++

```
class CSecurePrng : public MXD_CRYPTO_SECUREPRNG_CLASSNAME;
```

Description

CPrng is the class that manages a secure pseudo-random number generator.

Location

Crypto/CSecurePrng.h

Methods

Method	Description
Generate (see page 389)	Gets x random bytes.
SetSeed (see page 390)	Seeds the generator.

Legend



2.7.1.14.1 - Methods

2.7.1.14.1.1 - Generate

2.7.1.14.1.1.1 - CSecurePrng::Generate Method

Gets x random bytes.

C++

```
static mxt_result Generate(IN unsigned int uNumberOfBytes, OUT CBlob* pblobRandomBytes);
```

Parameters

Parameters	Description
IN unsigned int uNumberOfBytes	The number of random bytes to get.
OUT CBlob* pblobRandomBytes	The blob to contain the random data.

Returns

- resS_OK: the number has been set successfully
- resFE_FAIL: the generation failed.
- resFE_INVALID_ARGUMENT: pblobRandomBytes is NULL.
- resFE_NOT_IMPLEMENTED: the PRNG is not implemented.

Description

Generates pseudo-random bytes for use in cryptography (seeded at startup). Bytes are generated 20 at a time from the seed by using a SHA-1 hash of the last 20 bytes used and an internal counter. These bytes are regenerated every time the 20 previous bytes have been used by calls to GetRandomBytes.

2.7.1.14.1.1.2 - CSecurePrng::Generate Method

Gets x random bytes.

C++

```
static mxt_result Generate(IN unsigned int uNumberOfBytes, OUT uint8_t* puRandomBytes);
```

Parameters

Parameters	Description
IN unsigned int uNumberOfBytes	The number of random bytes to get.
OUT uint8_t* puRandomBytes	The byte array to contain the random data. This pointer cannot be NULL.

Returns

- resS_OK: the number has been set successfully
- resFE_FAIL: the generation failed.
- resFE_INVALID_ARGUMENT: puRandomBytes is NULL.
- resFE_NOT_IMPLEMENTED: the PRNG is not implemented.

Description

Generates pseudo-random bytes for use in cryptography (seeded at startup). Bytes are generated 20 at a time from the seed by using a Sha1Mac hash of the last 20 bytes, an internal counter, and the generate seed as the key. These bytes are regenerated every time the 20 previous bytes have been used by calls to GetRandomBytes.

2.7.1.14.1.2 - CSecurePrng::SetSeed Method

Seeds the generator.

C++

```
static mxt_result SetSeed(IN unsigned int uSeedSize, IN uint8_t* puSeedData);
```

Parameters

Parameters	Description
IN unsigned int uSeedSize	Number of bytes to seed the generator with.
IN uint8_t* puSeedData	The data to seed the generator.

Returns

- resS_OK: the number has been set successfully
- resFE_FAIL: the generation failed.
- resFE_INVALID_ARGUMENT: puSeedData is NULL.
- resFE_NOT_IMPLEMENTED: the PRNG is not implemented.

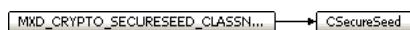
Description

Seeds the generator with x random bytes. These are meant to add additional entropy to the one already gathered to create the seed.

2.7.1.15 - CSecureSeed Class

Class used to generate random bytes to seed a pseudo-random generator.

Class Hierarchy



C++

```
class CSecureSeed : public MXD_CRYPTO_SECURESEED_CLASSNAME;
```

Description

Class used to generate random bytes to seed a pseudo-random generator. Once generated, these bytes are hashed by using SHA-1 to give a 20 byte seed.

Methods

Method	Description
GenerateSeed (see page 391)	Generates a seed.

Legend

	Method
--	--------

2.7.1.15.1 - Methods

2.7.1.15.1.1 - CSecureSeed::GenerateSeed Method

Generates a seed.

C++

```
static mxt_result GenerateSeed(OUT uint8_t* puSeedValue);
```

Parameters

Parameters	Description
OUT uint8_t* puSeedValue	Value of the generated seed.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT
- resFE_FAIL

Description

Generates a seed 20 bytes long. The seed is generated by using OS-specific data.

2.7.1.15.2 - Friends

2.7.1.15.2.1 - friend class Foo Friend

GNU 2.7.2 complains about private destructor with no friends.

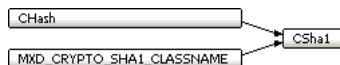
C++

```
friend class Foo;
```

2.7.1.16 - CSha1 Class

Class that manages the SHA-1 algorithm.

Class Hierarchy



C++

```
class CSha1 : public CHash, public MXD_CRYPTO_SHA1_CLASSNAME;
```

Description

CSha1 is the class that manages the SHA-1 algorithm.

Location

Crypto/CSha1.h

Constructors

Constructor	Description
• CSha1 (see page 392)	Constructor.

CHash Class

CHash Class	Description
• CHash (see page 352)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CSha1 (see page 393)	Destructor.

CHash Class

CHash Class	Description
• ~CHash (see page 352)	Destructor

Legend

•	Method
V	virtual

Methods

Method	Description
• V Begin (see page 393)	See CHash::Begin (see page 353).
• V End (see page 393)	Ends the hash.
• V GetAlgorithm (see page 394)	Retrieves the algorithm implemented within the inheriting class.
• V GetSizeInBits (see page 394)	Retrieves the size in bits of the hash returned by End.
• V GetSizeInBytes (see page 394)	Retrieves the size in bytes of the hash returned by End.
• V SetState (see page 394)	Sets the internal state from another instance.
• V Update (see page 395)	Updates the hash with new data.

CHash Class

CHash Class	Description
• A Begin (see page 353)	Begins a new hash.
• A End (see page 353)	Ends the hash.
• A GetAlgorithm (see page 354)	Retrieves the algorithm implemented within the inheriting class.
• A GetSizeInBits (see page 354)	Retrieves the size in bits of the hash returned by End (see page 353).
• A GetSizeInBytes (see page 354)	Retrieves the size in bytes of the hash returned by End (see page 353).
• A SetState (see page 354)	Sets the internal state from another instance.
• A Update (see page 355)	Updates the hash with new data.

Legend

•	Method
V	virtual
A	abstract

2.7.1.16.1 - Constructors

2.7.1.16.1.1 - CSha1::CSha1 Constructor

Constructor.

C++

```
CSha1();
```

Description

Default constructor.

2.7.1.16.2 - Destructors

2.7.1.16.2.1 - CSha1::~CSha1 Destructor

Destructor.

C++

```
virtual ~CSha1();
```

Description

Destructor.

2.7.1.16.3 - Methods

2.7.1.16.3.1 - CSha1::Begin Method

See CHash::Begin (See page 353).

C++

```
virtual mxt_result Begin(IN CHash::EAlgorithm eAlgorithm = CHash::eALGORITHM_DEFAULT);
```

2.7.1.16.3.2 - End

2.7.1.16.3.2.1 - CSha1::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.16.3.2.2 - CSha1::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.16.3.3 - CSha1::GetAlgorithm Method

Retrieves the algorithm implemented within the inheriting class.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

One of the CHash::EAlgorithm values.

Description

Retrieves the algorithm implemented within the inheriting class.

2.7.1.16.3.4 - CSha1::GetSizeInBits Method

Retrieves the size in bits of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size in bits.

Description

Retrieves the size in bits of the hash returned by End.

2.7.1.16.3.5 - CSha1::GetSizeInBytes Method

Retrieves the size in bytes of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size in bytes.

Description

Retrieves the size in bytes of the hash returned by End.

2.7.1.16.3.6 - CSha1::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHash* pHash);
```

Parameters

Parameters	Description
pHash	The other hash object instance from which to read the state. The hash algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.16.3.7 - Update

2.7.1.16.3.7.1 - CSha1::Update Method

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Updates the hash with new data. Begin must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.16.3.7.2 - CSha1::Update Method

See CHash::Update (see page 355).

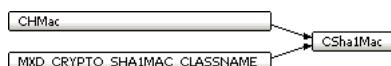
C++

```
virtual mxt_result Update(IN const uint8_t* puData, unsigned int uDataSize);
```

2.7.1.17 - CSha1Mac Class

Class that manages the SHA-1 Message Authentication Code algorithm.

Class Hierarchy



C++

```
class CSha1Mac : public CHMac, public MXD_CRYPTO_SHA1MAC_CLASSNAME;
```

Description

CSha1Mac is the class that manages the SHA-1 Message Authentication Code algorithm.

Location

Crypto/CSha1Mac.h

See Also

CSha1 (see page 391), CHMac (see page 355)

Constructors

Constructor	Description
• CSha1Mac (see page 396)	Constructor.

Legend



Destructors

Destructor	Description
• ~CSha1Mac (see page 396)	Destructor.

CHMac Class

CHMac Class	Description
~CHMac (see page 356)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
Begin (see page 397)	See CHMac::Begin (see page 356).
End (see page 397)	Ends the HMAC generation.
GetAlgorithm (see page 398)	Gets the algorithm.
GetSizelnBits (see page 398)	Gets the size in bits.
GetSizelnBytes (see page 398)	Gets the size in bytes.
SetState (see page 398)	Sets the internal state from another instance.
Update (see page 399)	Sets data to HMAC.

CHMac Class

CHMac Class	Description
Begin (see page 356)	Begins a new hash message authentication code.
End (see page 357)	Ends the HMAC generation.
GetAlgorithm (see page 358)	Gets the algorithm.
GetSizelnBits (see page 358)	Gets the size in bits.
GetSizelnBytes (see page 358)	Gets the size in bytes.
SetState (see page 359)	Sets the internal state from another instance.
Update (see page 359)	Sets data to HMAC.

Legend

	Method
	virtual
	abstract

2.7.1.17.1 - Constructors**2.7.1.17.1.1 - CSha1Mac::CSha1Mac Constructor**

Constructor.

C++

```
CSha1Mac();
```

Description

Default constructor.

2.7.1.17.2 - Destructors**2.7.1.17.2.1 - CSha1Mac::~CSha1Mac Destructor**

Destructor.

C++

```
virtual ~CSha1Mac();
```

Description

Destructor.

2.7.1.17.3 - Methods

2.7.1.17.3.1 - Begin

2.7.1.17.3.1.1 - CSha1Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const CBlob* pKey, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.17.3.1.2 - CSha1Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const uint8_t* puKey, IN unsigned int uKeySize, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.17.3.2 - End

2.7.1.17.3.2.1 - CSha1Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	Pointer to a blob to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.17.3.2.2 - CSha1Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	Pointer to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.17.3.3 - CSha1Mac::GetAlgorithm Method

Gets the algorithm.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

The algorithm used.

Description

Gets the algorithm used to create the hash.

2.7.1.17.3.4 - CSha1Mac::GetSizeInBits Method

Gets the size in bits.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size of the HMAC in bits

Description

Gets the size of the generated HMAC in bits.

2.7.1.17.3.5 - CSha1Mac::GetSizeInBytes Method

Gets the size in bytes.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size of the HMAC in bytes.

Description

Gets the size of the generated HMAC in bytes.

2.7.1.17.3.6 - CSha1Mac::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHMac* pHMac);
```

Parameters

Parameters	Description
pHash	The other hash-mac objet instance from which to read the state. The hash-mac algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.17.3.7 - Update

2.7.1.17.3.7.1 - CSha1Mac::Update Method

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Pointer to a blob containing the data.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End

2.7.1.17.3.7.2 - CSha1Mac::Update Method

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize);
```

Parameters

Parameters	Description
puData	Data to use to generate the HMAC.
uDataSize	Size of the data in bytes.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

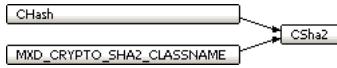
See Also

End

2.7.1.18 - CSha2 Class

Class that manages the SHA-2 algorithm.

Class Hierarchy



C++

```
class CSha2 : public CHash, public MXD_CRYPTO_SHA2_CLASSNAME;
```

Description

CSha2 is the class that manages the SHA-2 algorithm. The algorithm digest length is specified by the begin parameter. Only SHA-256 is supported for now.

Location

Crypto/CSha2.h

See Also

CHash (see page 351), CSha1 (see page 391)

Example

Here is a typical example of code used to do SHA-2 on the value puInData. The resulting digest value is returned in puHash.

```
void PerformHash(IN const uint8_t* puInData,
                  IN unsigned int uInDataSize,
                  OUT uint8_t* puHash)
{
    CSha2 sha2;
    sha2.Begin(CHash::eALGORITHM_SHA2_256);
    sha2.Update(puInData, uInDataSize);
    sha2.End(puHash);
}
```

Constructors

Constructor	Description
• CSha2 (see page 401)	Default Constructor.

CHash Class

CHash Class	Description
• CHash (see page 352)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CHash (see page 401)	Destructor.

CHash Class

CHash Class	Description
• ~CHash (see page 352)	Destructor

Legend

•	Method
▼	virtual

Methods

Method	Description
• Begin (see page 401)	See CHash::Begin (see page 353).
• End (see page 401)	Ends the hash.
• GetAlgorithm (see page 402)	Retrieves the algorithm implemented within the inheriting class.
• GetSizelnBits (see page 402)	Retrieves the size in bits of the hash returned by End.
• GetSizelnBytes (see page 402)	Retrieves the size in bytes of the hash returned by End.
• SetDefaultAlgorithm (see page 403)	Sets the default hashing algorithm to use when the eAlgorithm parameter of the Begin (see page 401) method is CHash::eALGORITHM_DEFAULT.
• SetState (see page 403)	Sets the internal state from another instance.
• Update (see page 403)	Updates the hash with new data.

CHash Class

CHash Class	Description
• Begin (see page 353)	Begins a new hash.
• End (see page 353)	Ends the hash.

• GetAlgorithm (see page 354)	Retrieves the algorithm implemented within the inheriting class.
• GetSizeInBits (see page 354)	Retrieves the size in bits of the hash returned by End (see page 353).
• GetSizeInBytes (see page 354)	Retrieves the size in bytes of the hash returned by End (see page 353).
• SetState (see page 354)	Sets the internal state from another instance.
• Update (see page 355)	Updates the hash with new data.

Legend

	Method
	virtual
	abstract

2.7.1.18.1 - Constructors

2.7.1.18.1.1 - CSha2::CSha2 Constructor

Default Constructor.

C++

```
CSha2();
```

2.7.1.18.2 - Destructors

2.7.1.18.2.1 - CSha2::~CSha2 Destructor

Destructor.

C++

```
virtual ~CSha2();
```

2.7.1.18.3 - Methods

2.7.1.18.3.1 - CSha2::Begin Method

See CHash::Begin (see page 353).

C++

```
virtual mxt_result Begin(IN CHash::EAlgorithm eAlgorithm = CHash::eALGORITHM_DEFAULT);
```

2.7.1.18.3.2 - End

2.7.1.18.3.2.1 - CSha2::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.18.3.2.2 - CSha2::End Method

Ends the hash.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	The output buffer to store the hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the hash. End must be called once to finalize the hash. It must be paired with a single call to Begin.

2.7.1.18.3.3 - CSha2::GetAlgorithm Method

Retrieves the algorithm implemented within the inheriting class.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

One of the CHash::EAlgorithm values.

Description

Retrieves the algorithm implemented within the inheriting class.

2.7.1.18.3.4 - CSha2::GetSizeInBits Method

Retrieves the size in bits of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size in bits.

Description

Retrieves the size in bits of the hash returned by End.

2.7.1.18.3.5 - CSha2::GetSizeInBytes Method

Retrieves the size in bytes of the hash returned by End.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size in bytes.

Description

Retrieves the size in bytes of the hash returned by End.

2.7.1.18.3.6 - CSha2::SetDefaultAlgorithm Method

Sets the default hashing algorithm to use when the eAlgorithm parameter of the Begin (see page 401) method is CHash::eALGORITHM_DEFAULT.

C++

```
virtual mxt_result SetDefaultAlgorithm(IN CHash::EAlgorithm eAlgorithm);
```

Parameters

Parameters	Description
IN CHash::EAlgorithm eAlgorithm	One of: CHash::eALGORITHM_SHA2_224, CHash (see page 351):eALGORITHM_SHA2_256, CHash::eALGORITHM_SHA2_384, CHash::eALGORITHM_SHA2_512.

Returns

resS_OK: Algorithm accepted. resFE_INVALID_ARGUMENT: eAlgorithm is not one of: CHash::eALGORITHM_SHA2_224, CHash (see page 351):eALGORITHM_SHA2_256, CHash::eALGORITHM_SHA2_384, CHash::eALGORITHM_SHA2_512.

Description

Sets the default hashing algorithm to use when the eAlgorithm parameter of the Begin (see page 401) method is CHash::eALGORITHM_DEFAULT.

2.7.1.18.3.7 - CSha2::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHash* pHash);
```

Parameters

Parameters	Description
pHash	The other hash object instance from which to read the state. The hash algorithm must be the same as the current object. A NULL parameter can be used to detect whether or not the method is implemented. If it is implemented, resS_OK is returned. Otherwise, it returns resFE_NOT_IMPLEMENTED.

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.18.3.8 - Update

2.7.1.18.3.8.1 - CSha2::Update Method

Updates the hash with new data.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Input data to hash.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Updates the hash with new data. Begin must have already been called once before Update is called. Update may be called more than once. Updating from a buffer one byte at a time gives the same hash as updating from the same buffer all at once.

2.7.1.18.3.8.2 - CSha2::Update Method

See CHash::Update (see page 355).

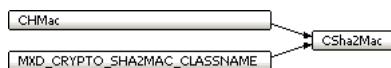
C++

```
virtual mxt_result Update(IN const uint8_t* puData, unsigned int uDataSize);
```

2.7.1.19 - CSha2Mac Class

Class that manages the SHA-2 Message Authentication Code algorithm.

Class Hierarchy



C++

```
class CSha2Mac : public CHMac, public MXD_CRYPTO_SHA2MAC_CLASSNAME;
```

Description

CSha2Mac is the class that manages the SHA-2 Message Authentication Code algorithm.

Location

Crypto/CSha2Mac.h

See Also

CSha2 (see page 399), CHMac (see page 355)

Constructors

Constructor	Description
• CSha2Mac (see page 405)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CSha2Mac (see page 405)	Destructor.

CHMac Class

CHMac Class	Description
• ~CHMac (see page 356)	Destructor.

Legend

•	Method
V	virtual

Methods

Method	Description
• Begin (see page 405)	See CHMac::Begin (see page 356).
• End (see page 406)	Ends the HMAC generation.
• GetAlgorithm (see page 406)	Gets the algorithm.
• GetSizeInBits (see page 406)	Gets the size in bits.
• GetSizeInBytes (see page 407)	Gets the size in bytes.
• SetDefaultAlgorithm (see page 407)	Sets the default digest bit length.
• SetState (see page 407)	Sets the internal state from another instance.
• Update (see page 407)	Sets data to HMAC.

CHMac Class

CHMac Class	Description
◆ A Begin (see page 356)	Begins a new hash message authentication code.
◆ A End (see page 357)	Ends the HMAC generation.
◆ A GetAlgorithm (see page 358)	Gets the algorithm.
◆ A GetSizeInBits (see page 358)	Gets the size in bits.
◆ A GetSizeInBytes (see page 358)	Gets the size in bytes.
◆ A SetState (see page 359)	Sets the internal state from another instance.
◆ A Update (see page 359)	Sets data to HMAC.

Legend

◆	Method
▼	virtual
▲	abstract

2.7.1.19.1 - Constructors

2.7.1.19.1.1 - CSha2Mac::CSha2Mac Constructor

Constructor.

C++

```
CSha2Mac();
```

Description

Basic constructor.

2.7.1.19.2 - Destructors

2.7.1.19.2.1 - CSha2Mac::~CSha2Mac Destructor

Destructor.

C++

```
virtual ~CSha2Mac();
```

Description

Destructor.

2.7.1.19.3 - Methods

2.7.1.19.3.1 - Begin

2.7.1.19.3.1.1 - CSha2Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const CBlob* pKey, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.19.3.1.2 - CSha2Mac::Begin Method

See CHMac::Begin (see page 356).

C++

```
virtual mxt_result Begin(IN const uint8_t* puKey, IN unsigned int uKeySize, IN CHMac::EAlgorithm eAlgorithm = CHMac::eALGORITHM_DEFAULT);
```

2.7.1.19.3.2 - End

2.7.1.19.3.2.1 - CSha2Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT CBlob* pHash);
```

Parameters

Parameters	Description
pHash	Pointer to a blob to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.19.3.2.2 - CSha2Mac::End Method

Ends the HMAC generation.

C++

```
virtual mxt_result End(OUT uint8_t* puHash);
```

Parameters

Parameters	Description
puHash	Pointer to contain the resulting HMAC.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Ends the generation of the HMAC. The generated HMAC is then returned.

See Also

End

2.7.1.19.3.3 - CSha2Mac::GetAlgorithm Method

Gets the algorithm.

C++

```
virtual EAlgorithm GetAlgorithm();
```

Returns

The algorithm used.

Description

Gets the algorithm used to create the hash.

2.7.1.19.3.4 - CSha2Mac::GetSizeInBits Method

Gets the size in bits.

C++

```
virtual unsigned int GetSizeInBits();
```

Returns

The size of the HMAC in bits

Description

Gets the size of the generated HMAC in bits.

2.7.1.19.3.5 - CSha2Mac::GetSizeInBytes Method

Gets the size in bytes.

C++

```
virtual unsigned int GetSizeInBytes();
```

Returns

The size of the HMAC in bytes.

Description

Gets the size of the generated HMAC in bytes.

2.7.1.19.3.6 - CSha2Mac::SetDefaultAlgorithm Method

Sets the default digest bit length.

C++

```
virtual mxt_result SetDefaultAlgorithm(IN CHMac::EAlgorithm eAlgorithm);
```

Parameters

Parameters	Description
IN CHMac::EAlgorithm eAlgorithm	The algorithm to use.

Returns

- resS_OK, the algorithm has correctly been set.
- Failure, the provided algorithm is unsupported.

Description

Sets the default digest bit length for the HMAC-SHA-2 algorithm.

2.7.1.19.3.7 - CSha2Mac::SetState Method

Sets the internal state from another instance.

C++

```
virtual mxt_result SetState(IN const CHMac* pHMac);
```

Returns

resS_OK: Method success. Other: Method failure.

Description

Sets the internal state from another instance.

2.7.1.19.3.8 - Update**2.7.1.19.3.8.1 - CSha2Mac::Update Method**

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const CBlob* pData);
```

Parameters

Parameters	Description
pData	Pointer to a blob containing the data.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End

2.7.1.19.3.8.2 - CSha2Mac::Update Method

Sets data to HMAC.

C++

```
virtual mxt_result Update(IN const uint8_t* puData, IN unsigned int uDataSize);
```

Parameters

Parameters	Description
puData	Data to use to generate the HMAC.
uDataSize	Size of the data in bytes.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Sets the specified data to HMAC. Multiple calls to the method result in the total data being hashed in blocks of the hash input size.

See Also

End

2.7.1.20 - IPassPhrase Class

Interface through which requests for pass phrases are done to decrypt an object stored in PEM format.

Class Hierarchy

IPassPhrase

C++

```
class IPassPhrase;
```

Description

This is the interface through which requests for pass phrases are done to to decrypt an object stored in PEM format.

Location

Crypto/IPassPhrase.h

Methods

Method	Description
  GetPassPhrase (See page 409)	Retrieves the pass phrase associated with a PEM object.

Legend

	Method
	abstract

2.7.1.20.1 - Methods

2.7.1.20.1.1 - IPassPhrase::GetPassPhrase Method

Retrieves the pass phrase associated with a PEM object.

C++

```
virtual mxt_result GetPassPhrase(IN mxt_opaque opqPassPhraseParameter, OUT CBlob* pPassPhrase) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqPassPhraseParameter	The parameter associated with the pass phrase context.
OUT CBlob* pPassPhrase	A container for the retrieved pass phrase.

Returns

An error if unable to provide the pass phrase.

Description

Called to retrieve the pass phrase associated with a PEM object. This pass phrase is used to decrypt the PEM object.

2.7.2 - Enumerations

This section documents the enumerations of the Sources/Crypto folder.

Enumerations

Enumeration	Description
EVariant (See page 409)	Types of variants.
EAction (See page 410)	The EAction enum specifies whether decryption or encryption should be performed. eACTION_DEFAULT specifies that the provisioned default action is to be used.
EAlgorithm (See page 410)	Enumerates all the supported cipher algorithms.
EMode (See page 410)	The EMode enum specifies the utilization mode of the cipher algorithm. What combination of fundamental cipher algorithm operations should be performed upon the input stream to generate the output stream. eMODE_DEFAULT specifies that the provisioned default mode is to be used.
ECrcType (See page 410)	CRC types.
EAlgorithm (See page 411)	Enumerates all the supported hash algorithms.
EAlgorithm (See page 411)	Enumerates all the supported HMac algorithms.

2.7.2.1 - CBase64::EVariant Enumeration

Types of variants.

C++

```
enum EVariant {
    eSTANDARD,
    eURL
};
```

Members

Members	Description
eSTANDARD	Standard variant.
eURL	URL variant.

2.7.2.2 - CCipher::EAction Enumeration

The EAction enum specifies whether decryption or encryption should be performed. eACTION_DEFAULT specifies that the provisioned default action is to be used.

C++

```
enum EAction {
    eACTION_DECRYPT,
    eACTION_ENCRYPT,
    eACTION_DEFAULT
};
```

Members

Members	Description
eACTION_DECRYPT	Decryption.
eACTION_ENCRYPT	Encryption.
eACTION_DEFAULT	Default action.

2.7.2.3 - CCipher::EAlgorithm Enumeration

Enumerates all the supported cipher algorithms.

C++

```
enum EAlgorithm {
    eALGORITHM_AES,
    eALGORITHM_BASE64
};
```

Members

Members	Description
eALGORITHM_AES	AES algorithm.
eALGORITHM_BASE64	Base 64 algorithm.

2.7.2.4 - CCipher::EMode Enumeration

The EMode enum specifies the utilization mode of the cipher algorithm. What combination of fundamental cipher algorithm operations should be performed upon the input stream to generate the output stream. eMODE_DEFAULT specifies that the provisioned default mode is to be used.

C++

```
enum EMode {
    eMODE_CBC,
    eMODE_CFB,
    eMODE_CTR,
    eMODE_ECB,
    eMODE_OFB,
    eMODE_DEFAULT
};
```

Members

Members	Description
eMODE_CBC	Cipher Block Chaining.
eMODE_CFB	Cipher Feedback.
eMODE_CTR	Counter.
eMODE_ECB	Electronic Code Book.
eMODE_OFB	Output Feedback.
eMODE_DEFAULT	Default mode.

2.7.2.5 - CCrc::ECrcType Enumeration

CRC types.

C++

```
enum ECrcType {
    eCRC16,
    eCRC32,
    eCRC32C
};
```

Description

Defines the possible types if CRC available.

Members

Members	Description
eCRC16	16 bits CRC CCITT.
eCRC32	32 bits CRC IEEE 802.3.
eCRC32C	32 bits CRC Castagnoli.

2.7.2.6 - CHash::EAlgorithm Enumeration

Enumerates all the supported hash algorithms.

C++

```
enum EAlgorithm {
    eALGORITHM_DEFAULT,
    eALGORITHM_MD5,
    eALGORITHM_SHA1,
    eALGORITHM_SHA2_224,
    eALGORITHM_SHA2_256,
    eALGORITHM_SHA2_384,
    eALGORITHM_SHA2_512
};
```

Members

Members	Description
eALGORITHM_DEFAULT	Default algorithm.
eALGORITHM_MD5	MD5 algorithm.
eALGORITHM_SHA1	SHA-1 algorithm.
eALGORITHM_SHA2_224	SHA-2 algorithm (224 bits).
eALGORITHM_SHA2_256	SHA-2 algorithm (256 bits).
eALGORITHM_SHA2_384	SHA-2 algorithm (384 bits).
eALGORITHM_SHA2_512	SHA-2 algorithm (512 bits).

2.7.2.7 - CHMac::EAlgorithm Enumeration

Enumerates all the supported HMac algorithms.

C++

```
enum EAlgorithm {
    eALGORITHM_DEFAULT = CHash::eALGORITHM_DEFAULT,
    eALGORITHM_MD5 = CHash::eALGORITHM_MD5,
    eALGORITHM_SHA1 = CHash::eALGORITHM_SHA1,
    eALGORITHM_SHA2_224 = CHash::eALGORITHM_SHA2_224,
    eALGORITHM_SHA2_256 = CHash::eALGORITHM_SHA2_256,
    eALGORITHM_SHA2_384 = CHash::eALGORITHM_SHA2_384,
    eALGORITHM_SHA2_512 = CHash::eALGORITHM_SHA2_512
};
```

Members

Members	Description
eALGORITHM_DEFAULT = CHash::eALGORITHM_DEFAULT	Default algorithm.
eALGORITHM_MD5 = CHash::eALGORITHM_MD5	MD5 algorithm.
eALGORITHM_SHA1 = CHash::eALGORITHM_SHA1	SHA-1 algorithm.
eALGORITHM_SHA2_224 = CHash::eALGORITHM_SHA2_224	SHA-2 algorithm (224 bits).
eALGORITHM_SHA2_256 = CHash::eALGORITHM_SHA2_256	SHA-2 algorithm (256 bits).
eALGORITHM_SHA2_384 = CHash::eALGORITHM_SHA2_384	SHA-2 algorithm (384 bits).
eALGORITHM_SHA2_512 = CHash::eALGORITHM_SHA2_512	SHA-2 algorithm (512 bits).

2.7.3 - Variables

This section documents the variables of the Sources/Crypto folder.

2.7.3.1 - uMD5_HASH_SIZE_IN_BITS Variable

```
const unsigned int uMD5_HASH_SIZE_IN_BITS = 128;
```

Description

Size of the MD5 hash in bits.

2.7.3.2 - uMD5_HASH_SIZE_IN_BYTES Variable

```
const unsigned int uMD5_HASH_SIZE_IN_BYTES = (uMD5_HASH_SIZE_IN_BITS / 8);
```

Description

Size of the MD5 hash in bytes.

2.7.3.3 - uMD5_MAC_SIZE_IN_BITS Variable

```
const unsigned int uMD5_MAC_SIZE_IN_BITS = 128;
```

Description

Size of the HMAC in bits.

2.7.3.4 - uMD5_MAC_SIZE_IN_BYTES Variable

```
const unsigned int uMD5_MAC_SIZE_IN_BYTES = (uMD5_MAC_SIZE_IN_BITS / 8);
```

Description

Size of the HMAC in bytes.

2.7.3.5 - uSHA1_HASH_SIZE_IN_BITS Variable

```
const unsigned int uSHA1_HASH_SIZE_IN_BITS = 160;
```

Description

The size of a SHA-1 hash in bits.

2.7.3.6 - uSHA1_HASH_SIZE_IN_BYTES Variable

```
const unsigned int uSHA1_HASH_SIZE_IN_BYTES = (uSHA1_HASH_SIZE_IN_BITS / 8);
```

Description

The size of a SHA-1 hash in bytes.

2.7.3.7 - uSHA256_BLOCK_SIZE Variable

```
const unsigned int uSHA256_BLOCK_SIZE = 64;
```

Description

The size of a SHA-256 block in bytes.

2.7.3.8 - uSHA256_HASH_SIZE_IN_BITS Variable

```
const unsigned int uSHA256_HASH_SIZE_IN_BITS = 256;
```

Description

The size of a SHA-256 hash in bits.

2.7.3.9 - uSHA256_HASH_SIZE_IN_BYTES Variable

```
const unsigned int uSHA256_HASH_SIZE_IN_BYTES = (uSHA256_HASH_SIZE_IN_BITS / 8);
```

Description

The size of a SHA-256 hash in bytes.

2.8 - ECom

This section documents the Sources/ECom folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 413)
- Functions (see page 420)

- Macros (see page 423)
- Types (see page 425)

2.8.1 - Classes

This section documents the classes of the Sources/ECom folder.

Classes

Class	Description
CEComDelegatingUnknown (see page 413)	Base class to be used when an ECOM (see page 412) class can be aggregated.
CEComUnknown (see page 414)	Used as the base class of every ECOM (see page 412) object when it is known that they will never be aggregated.
IEComUnknown (see page 416)	Base class to be inherited by all ECOM (see page 412) interfaces.

2.8.1.1 - CEComDelegatingUnknown Class

Base class to be used when an ECOM (see page 412) class can be aggregated.

Class Hierarchy



C++

```
class CEComDelegatingUnknown : public CEComUnknown;
```

Description

ECOM (see page 412) classes can be described from two points of view: the ECOM (see page 412) implementer's view and the ECOM (see page 412) user's view. Inheriting CEComDelegatingUnknown is related to the implementer's view.

This class MUST be used as the base class of every ECOM (see page 412) class when it is known that they can be aggregated. In addition to the behaviour provided by the CEComUnknown (see page 414) base class, this class holds another important information about the ECOM (see page 412) object's state: a pointer to the ECOM (see page 412) object's real IEComUnknown (see page 416) interface. It makes aggregation optional and transparent.

Any ECOM (see page 412) that inherits this class MUST:

- Implement the three IEComUnknown (see page 416) methods. It MUST always be made using the macro MX_DECLARE_DELEGATING_IECOMUNKNOWN (see page 423).
- Override at least NonDelegatingQueryIf to add the possibility to query for its supported interface(s). NonDelegatingAddIfRef and NonDelegatingReleaseIfRef can also be overridden if their default behaviour needs to be augmented.

See Also

CEComUnknown (see page 414), IEComUnknown (see page 416), IEComNonDelegatingUnknown

Constructors

Constructor	Description
CEComDelegatingUnknown (see page 414)	Constructor

CEComUnknown Class

CEComUnknown Class	Description
CEComUnknown (see page 415)	Constructor.

Legend



Destructors

Destructor	Description
~CEComDelegatingUnknown (see page 414)	Destructor

CEComUnknown Class

CEComUnknown Class	Description
~CEComUnknown (see page 416)	Destructor.

Legend

	Method
	virtual

Methods

CEComUnknown Class

CEComUnknown Class	Description
InitializeInstance (See page 416)	Initializes the instance.

Legend

	Method
	virtual

2.8.1.1.1 - Constructors

2.8.1.1.1.1 - CEComDelegatingUnknown::CEComDelegatingUnknown Constructor

Constructor

C++

```
CEComDelegatingUnknown( IN IEComUnknown* pOuterIEComUnknown = NULL );
```

Parameters

Parameters	Description
IN IEComUnknown* pOuterIEComUnknown = NULL	A pointer to the ECOM (See page 412) object aggregator. NULL if the ECOM (See page 412) object is not being aggregated.

Description

Constructor.

2.8.1.1.2 - Destructors

2.8.1.1.2.1 - CEComDelegatingUnknown::~CEComDelegatingUnknown Destructor

Destructor

C++

```
virtual ~CEComDelegatingUnknown();
```

Description

Destructor.

2.8.1.2 - CEComUnknown Class

Used as the base class of every ECOM (See page 412) object when it is known that they will never be aggregated.

Class Hierarchy



C++

```
class CEComUnknown : public IEComNonDelegatingUnknown;
```

Description

ECOM (See page 412) objects can be described from two points of view. The ECOM (See page 412) implementer's view and the ECOM (See page 412) user's view. Inheriting CEComUnknown is related to the implementer's view.

This class MUST be used as the base class of every ECOM (See page 412) object when it is known that they will never be aggregated. The class holds one important information about the object's state: the object's reference count. The class also provides the default behaviour for the three methods present in the IEComNonDelegatingUnknown interface.

Distinction between IEComUnknown and IEComNonDelegatingUnknown:

Every ECOM (see page 412) object inherits from both interfaces. IEComUnknown (see page 416) is the base interface of all the object's inherited interfaces and IEComNonDelegatingUnknown is the base interface of CEComUnknown also inherited by the object's.

IEComUnknown (see page 416) is the interface that is ALWAYS used by ECOM (see page 412) users. It is always implemented the same way. On the other hand, IEComNonDelegatingUnknown methods are NEVER used or invoked. IEComNonDelegatingUnknown is only a placeholder for ECOM (see page 412) implementers to override the specific IEComUnknown (see page 416) behaviour of their object.

Looking carefully at both interfaces reveals that they are almost identical. In fact, besides the names that differ, both interfaces have the same number of methods in the same order, each with the same number of arguments also in the same order. It has been voluntarily made this way to address aggregation more easily for ECOM (see page 412) implementers and make it transparent to ECOM (see page 412) users. When no aggregation is used, the ECOM (see page 412) abstract creation mechanism has the IEComUnknown (see page 416) interface point on the IEComNonDelegatingUnknown of the CEComUnknown class from which an ECOM (see page 412) inherits. Alternatively, when aggregation is used, the creation mechanism has the IEComUnknown (see page 416) interface point to the IEComNonDelegatingUnknown of the CEComUnknown inherited by the outer component. This mechanism is in place to address one of the QueryIf rules: "You always get the same IEComUnknown (see page 416)".

Any ECOM (see page 412) that inherits this class MUST:

- Implement the three IEComUnknown (see page 416) methods. It MUST always be made using the macro MX_DECLARE_IECOMUNKNOWN (see page 425).
- Override at least NonDelegatingQueryIf to add the possibility to query for its supported interface(s). NonDelegatingAddIfRef and NonDelegatingReleaseIfRef can also be overridden if their default behaviour needs to be augmented.

See Also

CEComDelegatingUnknown (see page 413), IEComUnknown (see page 416), IEComNonDelegatingUnknown

Constructors

Constructor	Description
CEComUnknown (see page 415)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CEComUnknown (see page 416)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
  InitializeInstance (see page 416)	Initializes the instance.

Legend

	Method
	virtual

2.8.1.2.1 - Constructors

2.8.1.2.1.1 - CEComUnknown::CEComUnknown Constructor

Constructor.

C++

```
CEComUnknown( IN IEComUnknown* pOuterIEComUnknown = NULL );
```

Parameters

Parameters	Description
IN IEComUnknown* pOuterIEComUnknown = NULL	A pointer to the outer IEComUnknown (see page 416) object if this object is being aggregated.

Description

Constructor.

2.8.1.2.2 - Destructors

2.8.1.2.2.1 - CEComUnknown::~CEComUnknown Destructor

Destructor.

C++

```
virtual ~CEComUnknown();
```

Description

Destructor.

2.8.1.2.3 - Methods

2.8.1.2.3.1 - CEComUnknown::InitializeInstance Method

Initializes the instance.

C++

```
virtual mxt_result InitializeInstance();
```

Returns

A mxt_result (see page 92) error code. This depends on the implementation of the initialize instance.

Description

Initializes information inside the ECOM (see page 412) instance.

See Also

UninitializeInstance, Reference counting rules (see page 416)

2.8.1.3 - IEComUnknown Class

Base class to be inherited by all ECOM (see page 412) interfaces.

Class Hierarchy

```
IEComUnknown
```

C++

```
class IEComUnknown;
```

Description

ECOM (see page 412) classes can be described from two points of view: the ECOM (see page 412) implementer's view and the ECOM (see page 412) user's view. IEComUnknown is the interface that is ALWAYS used by ECOM (see page 412) users.

This interface is at the base of the entire ECOM (see page 412) mechanism. It provides the capacity to dynamically discover the interfaces supported by an ECOM (see page 412) class with QueryIf (see page 418) and to manage the ECOM (see page 412) object's lifetime using reference counting with AddIfRef (see page 418) and ReleaseIfRef (see page 420).

Inheritance from this interface is limited to ECOM (see page 412) interfaces only. Moreover, ECOM (see page 412) interfaces MUST absolutely inherit from this interface. This restriction comes from the QueryIf (see page 418) rules described below.

Overriding IEComUnknown's methods is required by all ECOMs and MUST be done using MX_DECLARE_IECOMUNKNOWN (see page 425) or MX_DECLARE_DELEGATING_IECOMUNKNOWN (see page 423) macros.

QueryIf rules:

- You always get the same IEComUnknown.
- You can get an interface if you got it before.
- You can get the interface you have.
- You can always get back where you started.
- You can get there from anywhere if you can get there from somewhere.

Reference counting rules:

The reference count of an ECOM (see page 412) object determines if it must remain in memory. When the ECOM (see page 412) object is no longer needed, its reference count should become zero, causing it to delete itself from memory. Reference counting is achieved with AddRef (see page 418) and ReleaseRef (see page 420) to inform the ECOM (see page 412) object about the number of references kept on it.

- Always call AddRef (see page 418) in a method that returns an interface. QueryIf (see page 418) and CreateEComInstance (see page 420) are examples.
- Call AddRef (see page 418) each time a copy of an ECOM (see page 412) interface pointer is kept.
- Call ReleaseRef (see page 420) on an ECOM (see page 412) interface pointer before making it invalid.
- Keep the count right to avoid memory leaks. Each successful call to CreateEComInstance (see page 420), QueryIf (see page 418), and AddRef (see page 418) must be matched with an associated call to ReleaseRef (see page 420).

ECom object ownership rules:

- If you created the ECOM (see page 412) object, you're owning it.
- If you need an ECOM (see page 412) object, it doesn't necessarily mean you have to own it.
- If you own an ECOM (see page 412) object, this ECOM (see page 412) object should not own you.
- If an ECOM (see page 412) object's lifetime is longer than yours, you don't need to own it.
- A container owns its content like parents own their children, but the opposite is not true.

Warning

- In any ECOM (see page 412) class hierarchy, only one (1) occurrence of CEComUnknown (see page 414) or CEComDelegatingUnknown (see page 413) is permitted.
- In any ECOM (see page 412) class hierarchy, only one (1) implementation of each IEComUnknown interface method is permitted.
- It is an error for one ECOM (see page 412) to inherit from another ECOM (see page 412). In such a case, aggregation and containment can be used instead.

Location

ECom/IEComUnknown.h

See Also

CreateEComInstance (see page 420), CEComUnknown (see page 414), CEComDelegatingUnknown (see page 413)

Methods

Method	Description
◆ A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
◆ A QueryIf (see page 418)	Queries an object for a supported interface.
◆ A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.8.1.3.1 - Methods**2.8.1.3.1.1 - IEComUnknown::AddIfRef Method**

Increments the reference count on the ECOM (see page 412) implementing this interface.

C++

```
virtual unsigned int AddIfRef() = 0;
```

Returns

The increased reference count (meaningless see warning below!).

Description

This method is used to increment the reference count on the ECOM (see page 412) implementing this interface. It is the reference count that controls the lifetime of every ECOM (see page 412). See Reference counting rules (see page 416).

Warning

The return value is meaningless and is provided for debugging purposes only.

See Also

ReleaseIfRef (see page 420), Reference counting rules (see page 416)

2.8.1.3.1.2 - QueryIf**2.8.1.3.1.2.1 - IEComUnknown::QueryIf Method**

Queries an object for a supported interface.

C++

```
virtual mxt_result QueryIf(IN mxt_iid iidRequested, OUT void** ppInterface) = 0;
```

Parameters

Parameters	Description
IN mxt_iid iidRequested	The interface identifier (IID) of the interface to retrieve.
OUT void** ppInterface	The pointer of the interface pointer variable to store the requested interface.

Returns

- resS_OK: The query succeeded. ppInterface now contains a pointer to the requested interface.
- resFE_MITOSFW_ECOM_NOINTERFACE: No interface could be found corresponding to the iidRequested. In this condition, NULL is assigned the content of ppInterface.

Description

This method is used to query an object for a supported interface. If the object supports the interface specified by iidRequested, then the pointer to this interface is returned in ppInterface.

When successful, a call to QueryIf automatically increments the object's reference count.

```
// Example of QueryIf.
// Note the IID_ISomeOtherInterface parameter and the
// reinterpret_cast that are required.

ISomeOtherInterface* pSomeOtherInterface = ...
ISomeInterface* pSomeInterface = NULL;
mxt_result res =
    pSomeOtherInterface->
        QueryIf(IID_ISomeOtherInterface,
            reinterpret_cast<void**>(&pSomeInterface));
```

See Also

`mxt_iid` (see page 426), Reference counting rules (see page 416)

2.8.1.3.1.2.2 - IEComUnknown::QueryIf Method

Queries an object for a supported interface.

C++

```
template <class _Type> mxt_result QueryIf(OUT CSharedPtr<_Type>& rInterface);
```

Parameters

Parameters	Description
<code>OUT CSharedPtr<_Type>& rInterface</code>	Reference to the <code>CSharedPtr</code> (see page 75) object that will store the requested interface.

Returns

- `resS_OK`: The query succeeded. `rInterface` now contains a pointer to the requested interface.
- `resFE_MITOSFW_ECOM_NOINTERFACE`: No interface could be found corresponding to the `iidRequested`. In this condition, `NULL` is assigned the content of `ppInterface`.

Description

This is a templated version of the original `QueryIf` method. It is designed to ease code readability.

For this method to compile, the `rInterface` referenced class must contain `MX_DECLARE_ECOM_GETIID` (see page 424).

This method is used to query an object for a supported interface.

When successful, a call to `QueryIf` automatically increments the object's reference count.

```
// Example of QueryIf, using the templated method.

CSharedPtr<ISomeOtherInterface*> spSomeOtherInterface = ...
CSharedPtr<ISomeInterface> spSomeInterface;
mxt_result res = spSomeOtherInterface->QueryIf(spSomeInterface);
```

See Also

`MX_DECLARE_ECOM_GETIID` (see page 424), Reference counting rules (see page 416)

2.8.1.3.1.2.3 - IEComUnknown::QueryIf Method

Queries an object for a supported interface.

C++

```
template <class _Type> mxt_result QueryIf(OUT _Type** ppInterface);
```

Parameters

Parameters	Description
<code>OUT _Type** ppInterface</code>	The pointer of the interface pointer variable to store the requested interface.

Returns

- `resS_OK`: The query succeeded. `ppInterface` now contains a pointer to the requested interface.
- `resFE_MITOSFW_ECOM_NOINTERFACE`: No interface could be found corresponding to the `iidRequested`. In this condition, `NULL` is assigned the content of `ppInterface`.
- `resFE_INVALID_ARGUMENT`: `ppInterface` parameter is `NULL`.

Description

This is a templated version of the original `QueryIf` method. It is designed to ease code readability.

For this method to compile, the `**ppInterface` referenced class must contain `MX_DECLARE_ECOM_GETIID` (see page 424).

This method is used to query an object for a supported interface.

When successful, a call to `QueryIf` automatically increments the object's reference count.

```
// Example of QueryIf, using the templated method.

ISomeOtherInterface* pSomeOtherInterface = ...
ISomeInterface* pSomeInterface = NULL;
mxt_result res = pSomeOtherInterface->QueryIf(&pSomeInterface);
```

See Also

`MX_DECLARE_ECOM_GETIID` (see page 424), Reference counting rules (see page 416)

2.8.1.3.1.3 - `IEComUnknown::ReleaselfRef` Method

Decrements the reference count on the ECOM (see page 412) implementing this interface.

C++

```
virtual unsigned int ReleaseIfRef() = 0;
```

Returns

The decreased reference count (meaningless see warning below!).

Description

This method is used to decrement the reference count on the ECOM (see page 412) implementing this interface. It is the reference count that controls the lifetime of every ECOM (see page 412). When the reference count reaches zero, the ECOM (see page 412) deletes itself.

Warning

The return value is meaningless and is provided for debugging purposes only.

See Also

`AddIfRef` (see page 418), Reference counting rules (see page 416).

2.8.2 - Functions

This section documents the functions of the Sources/ECom folder.

Functions

Function	Description
<code>CreateEComInstance</code> (see page 420)	Creates an ECOM (see page 412) class instance.
<code>CreateEComInstance</code> (see page 421)	Creates an ECOM (see page 412) class instance.
<code>IsEqualECom</code> (see page 422)	Compares equality of two interfaces.
<code>IsEqualEComIID</code> (see page 422)	Compares equality of two interfaces identifier.
<code>RegisterECom</code> (see page 422)	Registers an ECOM (see page 412) class.
<code>UnregisterECom</code> (see page 423)	Unregisters an ECOM (see page 412) class.

2.8.2.1 - `CreateEComInstance` Function

Creates an ECOM (see page 412) class instance.

C++

```
mxt_result CreateEComInstance(IN const mxt_clsid clsid, IN IEComUnknown* pOuterIEComUnknown, IN const mxt_iid iidRequested, OUT void** ppInterface);
```

Parameters

Parameters	Description
<code>IN const mxt_clsid clsid</code>	The class identifier (CLSID) of the ECOM (see page 412) class to create.
<code>IN IEComUnknown* pOuterIEComUnknown</code>	The pointer to an outer ECOM (see page 412) class if the ECOM (see page 412) class to create is being aggregated. If non null, use <code>GetOwnerIEComUnknown</code> to assign argument.
<code>IN const mxt_iid iidRequested</code>	The interface identifier (IID) of the interface to retrieve.
<code>OUT void** ppInterface</code>	The pointer of the interface pointer variable to store the requested interface.

Returns

- `resS_OK`: The creation succeeded. `ppInterface` now contains a pointer to the requested interface.

- resFE_INVALID_ARGUMENT: The creation failed. The supplied ppInterface is invalid.
- resFE_MITOSFW_ECOM_NOINTERFACE: The creation failed. No interface has been found corresponding to the iidRequested. In this condition, NULL is assigned to the content of ppInterface
- resFE_MITOSFW_ECOM_CLSIDNOTAVAILABLE: The creation failed. The class identifier corresponding to clsid does not exist.
- Other(s): A result code that can be returned by the CEComUnknown::InitializeInstance (see page 416) of the ECOM (see page 412) class associated with the clsid if it was overridden.

Description

This method creates in memory an instance of ECOM (see page 412) class associated with clsid. Usually, IID_IComUnknown is assigned to iidRequested for the call, but it is not an obligation as long as it is supported by the class. The interface identified by iidRequested is assigned to ppInterface if supported by the ECOM (see page 412) class. The interface returned through ppInterface is the first reference and currently the unique one on the ECOM (see page 412) class.

When the ECOM (see page 412) class to create is not being aggregated, ppOuterIEComUnknown must be assigned NULL. Otherwise, it is called within another ECOM (see page 412) class and ppOuterIEComUnknown MUST be assigned with the ECOM (see page 412) class's GetOwnerIEComUnknown method. In that case, iidRequested MUST be assigned with IID_IComUnknown.

See Also

IEComUnknown (see page 416), mxt_clsid (see page 426), mxt_iid (see page 426), MX_DECLARE_ECOM_CLSID (see page 424), MX_DECLARE_ECOM_IID (see page 424)

2.8.2.2 - CreateEComInstance Function

Creates an ECOM (see page 412) class instance.

C++

```
template <class _Type> inline mxt_result CreateEComInstance(IN const mxt_clsid clsid, IN IComUnknown* pOuterIEComUnknown, OUT _Type** ppInterface);
```

Parameters

Parameters	Description
IN const mxt_clsid clsid	The class identifier (CLSID) of the ECOM (see page 412) class to create.
IN IComUnknown* pOuterIEComUnknown	The pointer to an outer ECOM (see page 412) class if the ECOM (see page 412) class to create is being aggregated. If non null, use GetOwnerIEComUnknown to assign argument.
OUT _Type** ppInterface	The pointer of the interface pointer variable to store the requested interface.

Returns

- resS_OK: The creation succeeded. ppInterface now contains a pointer to the requested interface.
- resFE_INVALID_ARGUMENT: The creation failed. The supplied ppInterface is invalid.
- resFE_MITOSFW_ECOM_NOINTERFACE: The creation failed. No interface has been found corresponding to the iidRequested. In this condition, NULL is assigned to the content of ppInterface
- resFE_MITOSFW_ECOM_CLSIDNOTAVAILABLE: The creation failed. The class identifier corresponding to clsid does not exist.
- Other(s): A result code that can be returned by the CEComUnknown::InitializeInstance (see page 416) of the ECOM (see page 412) class associated with the clsid if it was overridden.

Description

This is a templated version of the original CreateEComInstance method. It is designed to ease code readability.

This method creates in memory an instance of ECOM (see page 412) class associated with clsid. The interface is retrieved from the

ppInterface parameter and is returned if supported by the ECOM (see page 412) class. The interface returned through ppInterface is the first reference and currently the unique one on the ECOM (see page 412) class.

When the ECOM (see page 412) class to create is not being aggregated, ppOuterIEComUnknown must be assigned NULL. Otherwise, it is called within another ECOM (see page 412) class and ppOuterIEComUnknown must be assigned with the class's GetOwnerIEComUnknown method. In that case, ppInterface MUST be of type IEComUnknown (see page 416).

See Also

IEComUnknown (see page 416), mxt_clsid (see page 426), MX_DECLARE_ECOM_CLSID (see page 424), MX_DECLARE_ECOM_GETIID (see page 424), MX_DECLARE_ECOM_IID (see page 424)

2.8.2.3 - IsEqualECom Function

Compares equality of two interfaces.

C++

```
bool IsEqualECom(IN IEComUnknown* pIEComIf1, IN IEComUnknown* pIEComIf2);
```

Parameters

Parameters	Description
IN IEComUnknown* pIEComIf1	First interface.
IN IEComUnknown* pIEComIf2	Second interface.

Returns

- true: If pIEComIf1 is implemented by the same ECOM (see page 412) as pIEComIf2.
- false: If pIEComIf1 is not implemented by the same ECOM (see page 412) as pIEComIf2.

Description

This method is a simple utility that compares two interfaces and tells whether or not they are implemented by the same ECOM (see page 412) class.

2.8.2.4 - IsEqualEComIID Function

Compares equality of two interfaces identifier.

C++

```
inline bool IsEqualEComIID(IN const mxt_iid iid1, IN const mxt_iid iid2);
```

Parameters

Parameters	Description
IN const mxt_iid iid1	First interface identifier.
IN const mxt_iid iid2	Second interface identifier.

Returns

- true: If iid1 is equal to iid2.
- false: If iid1 is not equal to iid2.

Description

This method is a simple utility that compares two interfaces identifier and tells whether or not they are equal. It is really useful for overriding the QueryIf method.

See Also

IEComUnknown (see page 416), CEComUnknown (see page 414)

2.8.2.5 - RegisterECom Function

Registers an ECOM (see page 412) class.

C++

```
mxt_result RegisterECom(IN const mxt_clsid clsid, IN mxt_pfnCreateEComInstance pfnCreateEComInstance);
```

Parameters

Parameters	Description
IN const mxt_clsid clsid	The class identifier (CLSID).
IN mxt_pfnCreateEComInstance pfnCreateEComInstance	A function that serves to create a new instance.

Returns

- resS_OK: The registration succeeded.
- resFE_INVALID_ARGUMENT: The registration failed.
- resFE_INVALID_STATE: The unregistration failed - Already Exists.

Description

This method must be called to register an ECOM (see page 412) class before it can be created with CreateEComInstance (see page 420).

2.8.2.6 - UnregisterECom Function

Unregisters an ECOM (see page 412) class.

C++

```
mxt_result UnregisterECom(IN const mxt_clsid clsid);
```

Parameters

Parameters	Description
IN const mxt_clsid clsid	The class identifier (CLSID).

Returns

- resS_OK: The unregistration succeeded.
- resFE_INVALID_ARGUMENT: The unregistration failed.
- resFE_INVALID_STATE: The unregistration failed - Not Found.

Description

This method may be called to unregister an ECOM (see page 412) class. It is no longer instantiable from CreateEComInstance (see page 420).

2.8.3 - Macros

This section documents the macros of the Sources/ECom folder.

Macros

Macro	Description
MX_DECLARE_DELEGATING_IECOMUNKNOWN (see page 423)	Provides the unique implementation of the IEComUnknown (see page 416) interface.
MX_DECLARE_ECOM_CLSID (see page 424)	Declares an ECOM (see page 412) class identifier.
MX_DECLARE_ECOM_GETIID (see page 424)	Declares a method GetIID.
MX_DECLARE_ECOM_IID (see page 424)	Declares an ECOM (see page 412) interface identifier.
MX_DECLARE_IECOMUNKNOWN (see page 425)	Provides the unique implementation of the IEComUnknown (see page 416) interface.

2.8.3.1 - MX_DECLARE_DELEGATING_IECOMUNKNOWN Macro

Provides the unique implementation of the IEComUnknown (see page 416) interface.

C++

```
#define MX_DECLARE_DELEGATING_IECOMUNKNOWN template<class _Type> mxt_result QueryIf(OUT _Type** ppInterface) {  
    if (ppInterface != NULL) { return QueryIf((*ppInterface)->GetIID(), reinterpret_cast<void**>(ppInterface)); }  
    return resFE_INVALID_ARGUMENT; } template<class _Type> mxt_result QueryIf(OUT CSharedPtr<_Type>& rInterface)  
    _Type** ppInterface = static_cast<_Type**>(rInterface); return QueryIf((*ppInterface)->GetIID(),  
    reinterpret_cast<void**>(ppInterface)); } virtual mxt_result QueryIf(IN mxt_iid iidRequested, OUT void**
```

```
ppInterface) { return GetOwnerIEComUnknown()->QueryIf(iidRequested, ppInterface); } virtual unsigned int
AddIfRef() { return GetOwnerIEComUnknown()->AddIfRef(); } virtual unsigned int ReleaseIfRef() { return
GetOwnerIEComUnknown()->ReleaseIfRef(); }
```

Description

Provides the unique implementation of the IEComUnknown (see page 416) interface. As the macro shows, all calls to QueryIf are delegated to the IEComUnknown (see page 416) owner for this ECOM (see page 412).

See Also

CEComDelegatingUnknown (see page 413)

2.8.3.2 - MX_DECLARE_ECOM_CLSID Macro

Declares an ECOM (see page 412) class identifier.

C++

```
#define MX_DECLARE_ECOM_CLSID( szClassName ) const SEComGuid CLSID_##szClassName[ ] =
{ { sizeof(MX_MAKESTRING(szClassName)), MX_MAKESTRING(szClassName) } };
```

Parameters

Parameters	Description
szClassName	The name of the ECOM (see page 412) class used in the class identifier definition.

Description

Declares an ECOM (see page 412) class identifier ("CLSID") in the EComCLSID.h file of a package based on the class name.

The name of the variable is formed as follows: CLSID_ClassName. Its content is generated automatically and must be unique. Each ECOM (see page 412) inheriting from CEComUnknown (see page 414) or CEComDelegatingUnknown (see page 413) must have a CLSID assigned.

See Also

mxt_clsid (see page 426), CreateEComInstance (see page 420), CEComUnknown (see page 414), CEComDelegatingUnknown (see page 413)

2.8.3.3 - MX_DECLARE_ECOM_GETIID Macro

Declares a method GetIID.

C++

```
#define MX_DECLARE_ECOM_GETIID(szIfaceName) mxt_iid GetIID() const { return IID_##szIfaceName; }
```

Parameters

Parameters	Description
szIfaceName	The name of the interface used in the class definition

Description

Declares a method GetIID. This method is required to support the specialized versions of CreateEComInstance (see page 420) and QueryIf.

```
MX_DECLARE_ECOM_IID(ISomeInterface);

class ISomeInterface : public IEComUnknown
{
public:
    MX_DECLARE_ECOM_GETIID(ISomeInterface);

    ...
};
```

See Also

CreateEComInstance (see page 420), IEComUnknown::QueryIf (see page 418)

2.8.3.4 - MX_DECLARE_ECOM_IID Macro

Declares an ECOM (see page 412) interface identifier.

C++

```
#define MX_DECLARE_ECOM_IID(szIfaceName) const SEComGuid IID_##szIfaceName[ ] =
```

```
{{sizeof(MX_MAKESTRING(szInterfaceName)), MX_MAKESTRING(szInterfaceName) } };
```

Parameters

Parameters	Description
szInterfaceName	The name of the interface used in the class definition

Description

Declares an ECOM (see page 412) interface identifier ("IID") in the .h file of an ECOM (see page 412) interface based on the class name.

The name of the variable is formed as follows: IID_InterfaceName. Its content is generated automatically and must be unique. Each interface inheriting from IEComUnknown (see page 416) must have an IID assigned.

```
MX_DECLARE_ECOM_IID(ISomeInterface);

class ISomeInterface : public IEComUnknown
{
public:
    MX_DECLARE_ECOM_GETIID(ISomeInterface);

    ...
};
```

See Also

mxt_iid (see page 426), CreateEComInstance (see page 420), IEComUnknown::QueryIf (see page 418), IEComUnknown (see page 416)

2.8.3.5 - MX_DECLAREIECOMUNKNOWN Macro

Provides the unique implementation of the IEComUnknown (see page 416) interface.

C++

```
#define MX_DECLAREIECOMUNKNOWN template<class _Type> mxt_result QueryIf(OUT CSharedPtr<_Type>& rInterface) {
    _Type** ppInterface = static_cast<_Type**>(rInterface); return NonDelegatingQueryIf((*ppInterface)->GetIID(),
    reinterpret_cast<void**>(ppInterface)); } template<class _Type> mxt_result QueryIf(OUT _Type** ppInterface) { if
    (ppInterface != NULL) { return NonDelegatingQueryIf((*ppInterface)->GetIID(),
    reinterpret_cast<void**>(ppInterface)); } return resFE_INVALID_ARGUMENT; } virtual mxt_result QueryIf(IN mxt_iid
    iidRequested, OUT void** ppInterface) { return NonDelegatingQueryIf(iidRequested, ppInterface); } virtual
    unsigned int AddIfRef() { return NonDelegatingAddIfRef(); } virtual unsigned int ReleaseIfRef() { return
    NonDelegatingReleaseIfRef(); }
```

Description

Provides the unique implementation of the IEComUnknown (see page 416) interface. As the macro shows, all calls to QueryIf are delegated to the ECOM (see page 412) object itself.

Notes

Using a unique macro with the same implementation for MX_DECLAREIECOMUNKNOWN and MX_DECLAREDELEGATINGIECOMUNKNOWN (see page 423) would have been possible. The distinction was created for performance to avoid the cost of dereferencing the IEComUnknown (see page 416) owner when the ECOM (see page 412) object can not be aggregated.

See Also

CEComUnknown (see page 414)

2.8.4 - Types

This section documents the types of the Sources/ECom folder.

Types

Type	Description
mxt_clsid (see page 426)	ECOM (see page 412) class identifier basic data type. The CLSID is contained in a variable declared with the macro MX_DECLARE_ECOM_CLSID (see page 424) in each package EComCLSID.h include file.
mxt_iid (see page 426)	ECOM (see page 412) interface identifier basic data type. The IID is contained in a variable declared with the macro MX_DECLARE_ECOM_IID (see page 424) in each interface include file.
mxt_pfnCreateEComInstance (see page 426)	CreateEComInstance (see page 420) function prototype used by the ECOM (see page 412) mechanism to create a new instance of an ECOM (see page 412) class.

2.8.4.1 - mxt_clsid Type

ECOM (see page 412) class identifier basic data type. The CLSID is contained in a variable declared with the macro MX_DECLARE_ECOM_CLSID (see page 424) in each package EComCLSID.h include file.

C++

```
typedef const SEComGuid* mxt_clsid;
```

2.8.4.2 - mxt_iid Type

ECOM (see page 412) interface identifier basic data type. The IID is contained in a variable declared with the macro MX_DECLARE_ECOM_IID (see page 424) in each interface include file.

C++

```
typedef const SEComGuid* mxt_iid;
```

2.8.4.3 - mxt_pfnCreateEComInstance Type

CreateEComInstance (see page 420) function prototype used by the ECOM (see page 412) mechanism to create a new instance of an ECOM (see page 412) class.

C++

```
typedef mxt_result (* mxt_pfnCreateEComInstance)(IN IEComUnknown* pOuterIEComUnknown, OUT CEComUnknown** ppCEComUnknown);
```

2.9 - Kerberos

This section documents the Sources/Kerberos folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 426)
- Enumerations (see page 463)

2.9.1 - Classes

This section documents the classes of the Sources/Kerberos folder.

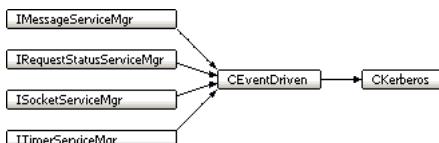
Classes

Class	Description
CKerberos (see page 426)	Core class that manages the Kerberos protocol.
CKerberosDomainToRealm (see page 432)	Container that stores two elements, a domain and a realm.
CKerberosGssContext (see page 435)	Base class for CKerberosGssContextAcceptor (see page 438) and CKerberosGssContextInitiator (see page 440).
CKerberosGssContextAcceptor (see page 438)	Implements the server side of a GSSAPI security context.
CKerberosGssContextInitiator (see page 440)	Implements the client side of a GSSAPI security context.
CKerberosGssContextOptions (see page 441)	Contains the state of all options related to a GSS context.
CKerberosPrincipal (see page 447)	Class representing a Kerberos principal.
CKerberosRealm (see page 452)	A container that stores information about a realm.
CKerberosTicket (see page 455)	A container that stores information about a ticket.
CKerberosTicketOptions (see page 456)	Contains the state of all options to use when requesting a ticket.
CKerberosTicketRequest (see page 461)	Class implementing Kerberos ticket request.
IKerberosTicketRequestMgr (see page 462)	This is the interface through which the kerberos ticket request reports its events.

2.9.1.1 - CKerberos Class

Core class that manages the Kerberos protocol.

Class Hierarchy



C++

```
class CKerberos : protected CEventDriven;
```

Description

CKerberos is the core class that manages the Kerberos protocol. It is the first class with which a user needs to interact.

What is Kerberos: Kerberos provides a means of verifying the identities of principals, (e.g. a workstation user or a network server) on an open (unprotected) network. This is accomplished without relying on assertions by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will. Kerberos performs authentication under these conditions as a trusted third-party authentication service by using conventional (shared secret key) cryptography.

The basic Kerberos authentication process proceeds as follows: A client sends a request to the authentication server (AS) requesting "credentials" for a given service. The AS responds with these credentials, encrypted in the client's key. The credentials consist of a "ticket" for the server and a temporary encryption key (often called a "session key"). The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the service's key) to the service. The session key (now shared by the client and server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

In order to add authentication to its transactions, a typical network application adds calls to the Kerberos library directly or through the Generic Security Services Application Programming Interface, GSSAPI. These calls result in the transmission of the necessary messages to achieve authentication.

The current implementation only supports authentication using GSSAPI, which is done over Kerberos.

Location

Kerberos/CKerberos.h

See Also

CKerberosGssContextAcceptor (see page 438), CKerberosGssContextInitiator (see page 440) CKerberosGssContextOptions (see page 441), CKerberosPrincipal (see page 447), CKerberosTicket (see page 455) CKerberosTicketOptions (see page 456), CKerberosTicketRequest (see page 461)

Constructors

Constructor	Description
CKerberos (see page 428)	Constructor.

CEventDriven Class

CEventDriven Class	Description
CEventDriven (see page 764)	Constructor.

Legend

	Method
--	--------

Destructors

CEventDriven Class

CEventDriven Class	Description
~CEventDriven (see page 764)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
GetGssContextAcceptor (see page 429)	Gets a new CKerberosGssContextAcceptor (see page 438) instance.
GetGssContextInitiator (see page 429)	Gets a new CKerberosGssContextInitiator (see page 440) instance.
GetGssContextOptions (see page 429)	Gets a new CKerberosGssContextOptions (see page 441) instance.
GetPrincipal (see page 430)	Gets a new CKerberosPrincipal (see page 447) instance.
GetTicketA (see page 430)	Requests a new ticket from the KDC.
GetTicketOptions (see page 430)	Gets a new CKerberosTicketOption instance.
Initialize (see page 431)	First initialization phase. Initializes basic kerberos service.
InitializeUser (see page 431)	Last initialization phase. Initializes user kerberos identity.
Release (see page 432)	Releases resources associated with the kerberos service.

CEventDriven Class

CEventDriven Class	Description
• Activate (see page 764)	Associates a Servicing Thread with this Event Driven.
• DisableCompletionDetection (see page 765)	Disables the detection of request completion.
• DisableEventsDetection (see page 765)	Disables the detection of events.
• EnableCompletionDetection (see page 766)	Enables the detection of request completion.
• EnableEventsDetection (see page 766)	Enables the detection of events.
• FinalizeAndReleaseA (see page 766)	Finalizes and releases an Event Driven.
• GetIEComUnknown (see page 766)	Returns a pointer to the Servicing Thread. AddRef is already called.
• GetServicingThread (see page 767)	Returns a pointer to the Servicing Thread. AddRef is already called.
• IsCurrentExecutionContext (see page 767)	Returns whether or not the code is executing within the current execution context.
• PostMessage (see page 767)	Pushes a new message into the message queue.
• RegisterRequestStatus (see page 768)	Registers a request status.
• RegisterSocket (see page 768)	Registers a socket.
• Release (see page 768)	Releases an Event Driven.
• StartTimer (see page 769)	Starts a new linear timer.
• StopAllTimers (see page 770)	Stops all timers owned by a manager.
• StopTimer (see page 770)	Stops a timer owned by a manager.
• UnregisterRequestStatus (see page 770)	Unregisters a request status.
• UnregisterSocket (see page 771)	Unregisters a socket.

ITimerServiceMgr Class

ITimerServiceMgr Class	Description
• A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

ISocketServiceMgr Class

ISocketServiceMgr Class	Description
• A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

IRequestStatusServiceMgr Class

IRequestStatusServiceMgr Class	Description
• A EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
A	abstract

2.9.1.1.1 - Constructors**2.9.1.1.1.1 - CKerberos::CKerberos Constructor**

Constructor.

C++

```
CKerberos();
```

Description

Constructor.

2.9.1.1.2 - Methods

2.9.1.1.2.1 - CKerberos::GetGssContextAcceptor Method

Gets a new CKerberosGssContextAcceptor (see page 438) instance.

C++

```
mxt_result GetGssContextAcceptor(OUT GO CKerberosGssContextAcceptor** ppContext);
```

Parameters

Parameters	Description
OUT GO CKerberosGssContextAcceptor** ppContext	On output, *ppContext contains a pointer to the newly allocated CKerberosGssContextAcceptor (see page 438) instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method returns a newly allocated CKerberosGssContextAcceptor (see page 438) instance in *ppContext.

2.9.1.1.2.2 - CKerberos::GetGssContextInitiator Method

Gets a new CKerberosGssContextInitiator (see page 440) instance.

C++

```
mxt_result GetGssContextInitiator(IN const CKerberosTicket* pServiceTicket, IN const CKerberosGssContextOptions* pOptions, OUT GO CKerberosGssContextInitiator** ppContext);
```

Parameters

Parameters	Description
IN const CKerberosTicket* pServiceTicket	A pointer to a CKerberosTicket (see page 455) used to identify with which service to initiate the GSS context. MUST NOT BE NULL.
IN const CKerberosGssContextOptions* pOptions	A pointer to a CKerberosGssContextOptions (see page 441) used to provide specific GSS context options. MAY BE NULL. If NULL, default options are used.
OUT GO CKerberosGssContextInitiator** ppContext	On output, *ppContext contains a pointer to the newly allocated CKerberosGssContextInitiator (see page 440) instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method returns a newly allocated CKerberosGssContextInitiator (see page 440) instance in *ppContext.

2.9.1.1.2.3 - CKerberos::GetGssContextOptions Method

Gets a new CKerberosGssContextOptions (see page 441) instance.

C++

```
mxt_result GetGssContextOptions(OUT GO CKerberosGssContextOptions** ppOptions);
```

Parameters

Parameters	Description
OUT GO CKerberosGssContextOptions** ppOptions	On output, *ppOptions contains a pointer to the newly allocated GetGssContextOptions instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT

- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method returns a newly allocated CKerberosGssContextOptions (see page 441) instance in *ppOptions.

2.9.1.1.2.4 - CKerberos::GetPrincipal Method

Gets a new CKerberosPrincipal (see page 447) instance.

C++

```
mxt_result GetPrincipal(OUT GO CKerberosPrincipal** ppPrincipal);
```

Parameters

Parameters	Description
OUT GO CKerberosPrincipal** ppPrincipal	On output, *ppPrincipal contains a pointer to the newly allocated CKerberosPrincipal (see page 447) instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method returns a newly allocated CKerberosPrincipal (see page 447) instance in *ppPrincipal.

2.9.1.1.2.5 - CKerberos::GetTicketA Method

Requests a new ticket from the KDC.

C++

```
mxt_result GetTicketA(IN mxt_opaque opq, IN IKerberosTicketRequestMgr* pMgr, IN const CKerberosPrincipal* pService, IN const CKerberosTicketOptions* pOptions, IN uint64_t uStartTimeMs, OUT GO CKerberosTicketRequest** ppRequest);
```

Parameters

Parameters	Description
IN mxt_opaque opq	Application-provided opaque data that will be later returned when the manager is notified.
IN IKerberosTicketRequestMgr* pMgr	The manager to notify in case of failure or success. MUST NOT be NULL.
IN const CKerberosPrincipal* pService	The service's principal name. NULL if the TGT is being requested.
IN const CKerberosTicketOptions* pOptions	A pointer to a CKerberosTicketOptions (see page 456) used to provided specific ticket options. MAY BE NULL. If NULL, default options will be used.
IN uint64_t uStartTimeMs	Specifies in MS the time remaining before the ticket is valid. This is used for postdated tickets.
OUT GO CKerberosTicketRequest** ppRequest	On output, *ppRequest contains a pointer to the newly allocated CKerberosTicketRequest (see page 461) instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method requests a new ticket for a specific service from the KDC. If pService is NULL, the TGT is retrieved. A CKerberosTicketRequest (see page 461) is returned if there is no error.

2.9.1.1.2.6 - CKerberos::GetTicketOptions Method

Gets a new CKerberosTicketOption instance.

C++

```
mxt_result GetTicketOptions(OUT GO CKerberosTicketOptions** ppOptions);
```

Parameters

Parameters	Description
OUT GO CKerberosTicketOptions** ppOptions	On output, *ppOptions contains a pointer to the newly allocated CKerberosTicketOptions (see page 456) instance. MUST NOT BE NULL.

Returns

- resS_OK
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method returns a newly allocated CKerberosTicketOptions (see page 456) instance in *ppOptions.

2.9.1.1.2.7 - CKerberos::Initialize Method

First initialization phase. Initializes basic kerberos service.

C++

```
mxt_result Initialize(IN IEComUnknown* pIEComUnknown, IN const CVector<CKerberosRealm>* pvecRealm, IN const CVector<CKerberosDomainToRealm>* pvecDomainToRealm = NULL);
```

Parameters

Parameters	Description
IN IEComUnknown* pIEComUnknown	A pointer to an interface serviced by a Servicing Thread.
IN const CVector<CKerberosRealm>* pvecRealm	A pointer to a vector of CKerberosRealms. It is used to configure the various realms accessible by the CKerberos (see page 426) user. Currently, the only information that may be configured is the IP address/port of the KDC. IMPORTANT: The first element is used as the default realm. It MUST NOT BE NULL and MUST NOT reference an empty vector.
IN const CVector<CKerberosDomainToRealm>* pvecDomainToRealm = NULL	A pointer to a vector of CKerberosDomainToRealms. It is used to map the host name to the realm for the KRB_NT_SRV_HST principal name type. It MAY BE NULL or MAY reference an empty vector.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This is the first method that must be called following the creation. It is used to initialize basic kerberos services.

2.9.1.1.2.8 - CKerberos::InitializeUser Method

Last initialization phase. Initializes user kerberos identity.

C++

```
mxt_result InitializeUser(IN const CKerberosPrincipal* pPrincipal, IN const CString* pstrPassword);
```

Parameters

Parameters	Description
IN const CKerberosPrincipal* pPrincipal	The user's principal name. MUST NOT BE NULL.
IN const CString* pstrPassword	A pointer to a CString (see page 126) that contains the user password used to decrypt the TGT. MUST NOT BE NULL.

Returns

- resS_OK

- resFE_FAIL
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method MUST be called following a call to Initialize (see page 431) and GetPrincipal (see page 430). It is used to initialize the user's kerberos identity. Required for authentication.

2.9.1.1.2.9 - CKerberos::Release Method

Releases resources associated with the kerberos service.

C++

```
void Release();
```

Description

Releases the resource associated with the kerberos service.

2.9.1.2 - CKerberosDomainToRealm Class

Container that stores two elements, a domain and a realm.

Class Hierarchy

CKerberosDomainToRealm

C++

```
class CKerberosDomainToRealm;
```

Description

Container that stores two elements, a domain and a realm. The purpose of this container is the mapping of a domain to a realm.

Location

Kerberos/CKerberosDomainToRealm.h

Constructors

Constructor	Description
CKerberosDomainToRealm (see page 433)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CKerberosDomainToRealm (see page 433)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
GetDomain (see page 433)	Gets the domain part.
GetRealm (see page 434)	Gets the realm part.
SetDomain (see page 434)	Sets the domain part.
SetRealm (see page 434)	Sets the realm part.

Legend

	Method
---	--------

2.9.1.2.1 - Constructors

2.9.1.2.1.1 - CKerberosDomainToRealm

2.9.1.2.1.1.1 - CKerberosDomainToRealm::CKerberosDomainToRealm Constructor

Default constructor.

C++

```
CKerberosDomainToRealm();
```

Description

Constructor.

2.9.1.2.1.1.2 - CKerberosDomainToRealm::CKerberosDomainToRealm Constructor

Copy constructor.

C++

```
CKerberosDomainToRealm(IN const CKerberosDomainToRealm& rDomainToRealm);
```

Parameters

Parameters	Description
IN const CKerberosDomainToRealm& rDomainToRealm	A reference to a CKerberosDomainToRealm.

Description

Copy constructor.

2.9.1.2.1.1.3 - CKerberosDomainToRealm::CKerberosDomainToRealm Constructor

Constructor. Built with domain and realm string.

C++

```
CKerberosDomainToRealm(IN const CString* pstrDomain, IN const CString* pstrRealm);
```

Parameters

Parameters	Description
IN const CString* pstrDomain	A pointer to a CString (see page 126) holding the domain.
IN const CString* pstrRealm	A pointer to a CString (see page 126) holding the realm.

Description

Constructor. Built with domain and realm string.

2.9.1.2.2 - Destructors

2.9.1.2.2.1 - CKerberosDomainToRealm::~CKerberosDomainToRealm Destructor

Destructor.

C++

```
virtual ~CKerberosDomainToRealm();
```

Description

Destructor.

2.9.1.2.3 - Methods

2.9.1.2.3.1 - CKerberosDomainToRealm::GetDomain Method

Gets the domain part.

C++

```
mxt_result GetDomain(OUT CString* pstrDomain) const;
```

Parameters

Parameters	Description
OUT CString* pstrDomain	Pointer to a CString (see page 126) to hold the domain.

Returns

resS_OK resFE_FAIL

Description

Gets the domain stored in this CKerberosDomainToRealm (see page 432) object.

2.9.1.2.3.2 - CKerberosDomainToRealm::GetRealm Method

Gets the realm part.

C++

```
mxt_result GetRealm(OUT CString* pstrRealm) const;
```

Parameters

Parameters	Description
OUT CString* pstrRealm	Pointer to a CString (see page 126) to hold the realm.

Returns

resS_OK resFE_FAIL

Description

Gets the realm stored in this CKerberosDomainToRealm (see page 432) object.

2.9.1.2.3.3 - CKerberosDomainToRealm::SetDomain Method

Sets the domain part.

C++

```
mxt_result SetDomain(IN const CString* pstrDomain);
```

Parameters

Parameters	Description
IN const CString* pstrDomain	Pointer to a CString (see page 126) that contains the domain.

Returns

resS_OK resFE_FAIL

Description

Sets the domain to store in this CKerberosDomainToRealm (see page 432) object.

2.9.1.2.3.4 - CKerberosDomainToRealm::SetRealm Method

Sets the realm part.

C++

```
mxt_result SetRealm(IN const CString* pstrRealm);
```

Parameters

Parameters	Description
IN const CString* pstrRealm	Pointer to a CString (see page 126) that contains the realm.

Returns

resS_OK resFE_FAIL

Description

Sets the realm to store in this CKerberosDomainToRealm (see page 432) object.

2.9.1.3 - CKerberosGssContext Class

Base class for CKerberosGssContextAcceptor (see page 438) and CKerberosGssContextInitiator (see page 440).

Class Hierarchy

 CKerberosGssContext

C++

```
class CKerberosGssContext;
```

Description

This class is the base class for CKerberosGssContextAcceptor (see page 438) and CKerberosGssContextInitiator (see page 440). It provides combined functionalities to both inheriting classes. Three different features are provided: message integrity, confidentiality, and resource releasing.

Message integrity should be thought of as the equivalent of a HMAC function. Confidentiality should be thought of as the equivalent of a cipher function.

Location

Kerberos/CKerberosGssContext.h

See Also

CKerberosGssContextAcceptor (see page 438), CKerberosGssContextInitiator (see page 440)

Destructors

Destructor	Description
 ~CKerberosGssContext (see page 435)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 GetMix (see page 436)	Generates a message integrity token for a specific message.
 Release (see page 436)	Releases resources associated with the GSS security context.
 Unwrap (see page 436)	Generates a message from a confidentiality token. Decrypts the token and returns a message.
 VerifyMix (see page 437)	Verifies that a message integrity token is valid for a specific message.
 Wrap (see page 437)	Generates a confidentiality token for a specific message. Encrypts the message and returns a token.

Legend

	Method
--	--------

2.9.1.3.1 - Destructors

2.9.1.3.1.1 - CKerberosGssContext::~CKerberosGssContext Destructor

Destructor.

C++

```
virtual ~CKerberosGssContext();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a

CKerberosGssContext (see page 435) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.9.1.3.2 - Methods

2.9.1.3.2.1 - CKerberosGssContext::GetMix Method

Generates a message integrity token for a specific message.

C++

```
mxt_result GetMix(IN const CBlob* pMessage, OUT CBlob* pToken) const;
```

Parameters

Parameters	Description
IN const CBlob* pMessage	The message. It MUST NOT be NULL.
OUT CBlob* pToken	On output, contains the generated message integrity token. It MUST NOT be NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_GSS_BAD_QOP
- resFE_MITOSFW_KERBEROS_GSS_CONTEXT_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

Generates a message integrity token for a specific message.

2.9.1.3.2.2 - CKerberosGssContext::Release Method

Releases resources associated with the GSS security context.

C++

```
void Release();
```

Description

Releases resources associated with the GSS security context.

2.9.1.3.2.3 - CKerberosGssContext::Unwrap Method

Generates a message from a confidentiality token. Decrypts the token and returns a message.

C++

```
mxt_result Unwrap(IN const CBlob* pToken, OUT CBlob* pMessage) const;
```

Parameters

Parameters	Description
IN const CBlob* pToken	The received token that must be decrypted. It MUST NOT be NULL.
OUT CBlob* pMessage	On output, contains the decrypted message. It MUST NOT be NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_GSS_BAD_SIG

- resFE_MITOSFW_KERBEROS_GSS_CONTEXT_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_DUPLICATE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_GAP_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_GSS_OLD_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_UNSEQ_TOKEN
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

Generates a message from a confidentiality token. Decrypts the token and returns a message.

2.9.1.3.2.4 - CKerberosGssContext::VerifyMix Method

Verifies that a message integrity token is valid for a specific message.

C++

```
mxt_result VerifyMix(IN const CBlob* pMessage, IN const CBlob* pToken) const;
```

Parameters

Parameters	Description
IN const CBlob* pMessage	The message. It MUST NOT be NULL.
IN const CBlob* pToken	The received token. It MUST NOT be NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_GSS_BAD_SIG
- resFE_MITOSFW_KERBEROS_GSS_CONTEXT_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_DUPLICATE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_GAP_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_GSS_OLD_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_UNSEQ_TOKEN
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

Verifies that a message integrity token is valid for a specific message.

2.9.1.3.2.5 - CKerberosGssContext::Wrap Method

Generates a confidentiality token for a specific message. Encrypts the message and returns a token.

C++

```
mxt_result Wrap(IN const CBlob* pMessage, OUT CBlob* pToken) const;
```

Parameters

Parameters	Description
IN const CBlob* pMessage	The message that must be encrypted.
OUT CBlob* pToken	On output, contains the encrypted token. It MUST NOT be NULL.

Returns

- resS_OK
- resFE_FAIL
- resFE_MITOSFW_KERBEROS_GSS_BAD_QOP
- resFE_MITOSFW_KERBEROS_GSS_CONTEXT_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

Generates a confidentiality token for a specific message. Encrypts the message and returns a token.

2.9.1.4 - CKerberosGssContextAcceptor Class

Implements the server side of a GSSAPI security context.

Class Hierarchy



C++

```
class CKerberosGssContextAcceptor : public CKerberosGssContext;
```

Description

The class CKerberosGssContextAcceptor implements the server side of a GSSAPI security context. It offers all the functionalities of a CKerberosGssContext (see page 435). In addition, it offers the added functionality of being able to establish a server security context. This is accomplished by calling the method Accept (see page 439).

Location

Kerberos/CKerberosGssContextAcceptor.h

See Also

CKerberosGssContextInitiator (see page 440)

Destructors

CKerberosGssContext Class

CKerberosGssContext Class	Description
~CKerberosGssContext (see page 435)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
Accept (see page 439)	Accepts a GSSAPI security context.

CKerberosGssContext Class

CKerberosGssContext Class	Description
GetMix (see page 436)	Generates a message integrity token for a specific message.
Release (see page 436)	Releases resources associated with the GSS security context.
Unwrap (see page 436)	Generates a message from a confidentiality token. Decrypts the token and returns a message.

• VerifyMix (see page 437)	Verifies that a message integrity token is valid for a specific message.
• Wrap (see page 437)	Generates a confidentiality token for a specific message. Encrypts the message and returns a token.

Legend

•	Method
---	--------

2.9.1.4.1 - Methods

2.9.1.4.1.1 - CKerberosGssContextAcceptor::Accept Method

Accepts a GSSAPI security context.

C++

```
mxt_result Accept(IN const CBlob* pIn, OUT CBlob* pOut, OUT CKerberosGssContextOptions** ppEffectiveOptions, OUT CKerberosPrincipal** ppClient);
```

Parameters

Parameters	Description
IN const CBlob* pIn	A token that has been received from the initiator. It MUST NOT be NULL.
OUT CBlob* pOut	On output, may contain a token that must be sent to the initiator. A token must be sent if the pOut blob is not empty. It MUST NOT be NULL.
OUT CKerberosGssContextOptions** ppEffectiveOptions	On output, contains the security context options that have been chosen. It MAY be NULL.
OUT CKerberosPrincipal** ppClient	On output, contains the identity of the client that initiated the security context. It MAY be NULL.

Returns

- resS_OK
- resFE_FAIL
- resS_MITOSFW_KERBEROS_GSS_COMPLETE
- resS_MITOSFW_KERBEROS_GSS_CONTINUE_NEEDED
- resFE_MITOSFW_KERBEROS_GSS_BAD_BINDINGS
- resFE_MITOSFW_KERBEROS_GSS_BAD_MECH
- resFE_MITOSFW_KERBEROS_GSS_BAD_SIG
- resFE_MITOSFW_KERBEROS_GSS_CREDENTIALS_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_CREDENTIAL
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_DUPLICATE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_GSS_NO_CRED
- resFE_MITOSFW_KERBEROS_GSS_OLD_TOKEN
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method accepts a GSSAPI security context. It must be called until it returns resS_MITOSFW_KERBEROS_GSS_COMPLETE or an error is returned. Except for the first call, every other call should be done after receiving an incoming token.

The following is the description of the most important result codes:

- resS_MITOSFW_KERBEROS_GSS_COMPLETE: Indicates that the security context has been successfully accepted. If the returned pOut token is not empty, it provides sufficient information for the initiator to perform message integrity and confidentiality.
- resS_MITOSFW_KERBEROS_GSS_CONTINUE_NEEDED: Indicates that the pOut token, must be sent to the initiator and that a reply must be received and passed back to Accept as the pln argument. In other words, it is possible for Accept to be called more

than once before `resS_MITOSFW_KERBEROS_GSS_COMPLETE` is finally received.

Warning

`pOut` may still contain a valid token to be sent, even in case of an error. If a valid token is present and an error is returned, the user must make sure that it gets sent to the initiator. This token could contain an encapsulated error that the initiator needs to receive.

2.9.1.5 - CKerberosGssContextInitiator Class

Implements the client side of a GSSAPI security context.

Class Hierarchy



C++

```
class CKerberosGssContextInitiator : public CKerberosGssContext;
```

Description

The class `CKerberosGssContextInitiator` implements the client side of a GSSAPI security context. It offers all the functionalities of a `CKerberosGssContext` (see page 435). In addition, it offers the added functionality of being able to establish a client security context. This is accomplished by calling the method `Initiate` (see page 440).

Location

`Kerberos/CKerberosGssContextInitiator.h`

See Also

`CKerberosGssContextAcceptor` (see page 438)

Destructors

CKerberosGssContext Class

CKerberosGssContext Class	Description
~CKerberosGssContext (see page 435)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
Initiate (see page 440)	Initiates a GSSAPI security context.

CKerberosGssContext Class

CKerberosGssContext Class	Description
GetMix (see page 436)	Generates a message integrity token for a specific message.
Release (see page 436)	Releases resources associated with the GSS security context.
Unwrap (see page 436)	Generates a message from a confidentiality token. Decrypts the token and returns a message.
VerifyMix (see page 437)	Verifies that a message integrity token is valid for a specific message.
Wrap (see page 437)	Generates a confidentiality token for a specific message. Encrypts the message and returns a token.

Legend

◆	Method
---	--------

2.9.1.5.1 - Methods

2.9.1.5.1.1 - CKerberosGssContextInitiator::Initiate Method

Initiates a GSSAPI security context.

C++

```
mxt_result Initiate(IN const CBlob* pIn, OUT CBlob* pOut, OUT CKerberosGssContextOptions** ppEffectiveOptions);
```

Parameters

Parameters	Description
IN const CBlob* pIn	A token received from the acceptor. It MAY be NULL or empty.
OUT CBlob* pOut	On output, may contain a token that must be sent to the acceptor. A token must be sent if the pOut blob is not empty. It MUST NOT be NULL.
OUT CKerberosGssContextOptions** ppEffectiveOptions	On output, contains the security context options that have been chosen. It MAY be NULL.

Returns

- resS_OK
- resFE_FAIL
- resS_MITOSFW_KERBEROS_GSS_COMPLETE
- resS_MITOSFW_KERBEROS_GSS_CONTINUE_NEEDED
- resFE_MITOSFW_KERBEROS_GSS_BAD_BINDINGS
- resFE_MITOSFW_KERBEROS_GSS_BAD_MECH
- resFE_MITOSFW_KERBEROS_GSS_BAD_NAME
- resFE_MITOSFW_KERBEROS_GSS_BAD_SIG
- resFE_MITOSFW_KERBEROS_GSS_CREDENTIALS_EXPIRED
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_CREDENTIAL
- resFE_MITOSFW_KERBEROS_GSS_DEFECTIVE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_DUPLICATE_TOKEN
- resFE_MITOSFW_KERBEROS_GSS_FAILURE
- resFE_MITOSFW_KERBEROS_GSS_NO_CONTEXT
- resFE_MITOSFW_KERBEROS_GSS_NO_CRED
- resFE_MITOSFW_KERBEROS_GSS_OLD_TOKEN
- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE

Description

This method initiates a GSSAPI security context. It must be called until it returns resS_MITOSFW_KERBEROS_GSS_COMPLETE or an error is returned. Except for the first call, every other call should be done after receiving an incoming token.

The following is the description of the most important result codes:

- resS_MITOSFW_KERBEROS_GSS_COMPLETE: Indicates that the security context has been successfully initiated. If the returned pOut token is not empty, it provides sufficient information for the acceptor to perform message integrity and confidentiality.
- resS_MITOSFW_KERBEROS_GSS_CONTINUE_NEEDED: Indicates that the pOut token must be sent to the acceptor and that a reply must be received and passed back to Initiate as the pln argument. In other words, it is possible for Initiate to be called more than once before resS_MITOSFW_KERBEROS_GSS_COMPLETE is finally received.

Warning

pOut may still contain a valid token to be sent even in case of an error. If a valid token is present and an error is returned, the user must make sure that it gets sent to the acceptor. This token could contain an encapsulated error that the acceptor needs to receive.

2.9.1.6 - CKerberosGssContextOptions Class

Contains the state of all options related to a GSS context.

Class Hierarchy

[CKerberosGssContextOptions](#)

C++

```
class CKerberosGssContextOptions;
```

Description

CKerberosGssContextOptions contains the state of all options related to a GSS context.

Location

Kerberos/CKerberosGssContextOptions.h

Destructors

Destructor	Description
 ~CKerberosGssContextOptions (see page 442)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 CloneFrom (see page 442)	Clones this object from the parameter.
 GetAnonymous (see page 443)	Gets the is anonymous flag.
 GetConfidentiality (see page 443)	Gets the confidentiality flag.
 GetCredentialDelagation (see page 443)	Gets the credential delegation flag.
 GetIntegrity (see page 444)	Gets the integrity flag.
 GetMutualAuthentication (see page 444)	Gets the mutual authentication flag.
 GetOutOfSequenceDetection (see page 444)	Gets the out of sequence detection flag.
 GetReplayDetection (see page 445)	Gets the replay detection flag.
 Release (see page 445)	Releases the Gcc context options object.
 SetAnonymous (see page 445)	Sets the is anonymous flag.
 SetConfidentiality (see page 445)	Sets the confidentiality flag.
 SetCredentialDelagation (see page 446)	Sets the credential delegation flag.
 SetIntegrity (see page 446)	Sets the integrity flag.
 SetMutualAuthentication (see page 446)	Sets the mutual authentication flag.
 SetOutOfSequenceDetection (see page 447)	Sets the out of sequence detection flag.
 SetReplayDetection (see page 447)	Sets the replay detection flag.

Legend

	Method
---	--------

2.9.1.6.1 - Destructors

2.9.1.6.1.1 - CKerberosGssContextOptions::~CKerberosGssContextOptions Destructor

Destructor.

C++

```
virtual ~CKerberosGssContextOptions();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a CKerberosGssContextOptions (see page 441) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.9.1.6.2 - Methods

2.9.1.6.2.1 - CKerberosGssContextOptions::CloneFrom Method

Clones this object from the parameter.

C++

```
mxt_result CloneFrom( IN const CKerberosGssContextOptions* pOptions);
```

Parameters

Parameters	Description
IN const CKerberosGssContextOptions* pOptions	Pointer to a CKerberosGssContextOptions object.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE
- resS_OK

Description

Clones the context options from another CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.2 - CKerberosGssContextOptions::GetAnonymousity Method

Gets the is anonymous flag.

C++

```
mxt_result GetAnonymousity(OUT bool* pbAnonymous);
```

Parameters

Parameters	Description
OUT bool* pbAnonymous	Pointer to hold the Anonymous flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the anonymous flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.3 - CKerberosGssContextOptions::GetConfidentiality Method

Gets the confidentiality flag.

C++

```
mxt_result GetConfidentiality(OUT bool* pbConfidential);
```

Parameters

Parameters	Description
OUT bool* pbConfidential	Pointer to hold the confidentiality flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the confidentiality flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.4 - CKerberosGssContextOptions::GetCredentialDelagation Method

Gets the credential delegation flag.

C++

```
mxt_result GetCredentialDelagation(OUT bool* pbDelegateConfidential);
```

Parameters

Parameters	Description
OUT bool * pbDelegateCndential	Pointer to hold the credential delegation flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the credential delegation flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.5 - CKerberosGssContextOptions::GetIntegrity Method

Gets the integrity flag.

C++

```
mxt_result GetIntegrity(OUT bool* pbIntegrity);
```

Parameters

Parameters	Description
OUT bool * pbIntegrity	Pointer to hold the integrity flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the integrity flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.6 - CKerberosGssContextOptions::GetMutualAuthentication Method

Gets the mutual authentication flag.

C++

```
mxt_result GetMutualAuthentication(OUT bool* pbMutualAuthentication);
```

Parameters

Parameters	Description
OUT bool * pbMutualAuthentication	Pointer to hold the mutual authentication flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the mutual authentication flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.7 - CKerberosGssContextOptions::GetOutOfSequenceDetection Method

Gets the out of sequence detection flag.

C++

```
mxt_result GetOutOfSequenceDetection(OUT bool* pbOutOfSequenceDetection);
```

Parameters

Parameters	Description
OUT bool * pbOutOfSequenceDetection	Pointer to hold the out of sequence detection flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the out of sequence detection flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.8 - CKerberosGssContextOptions::GetReplayDetection Method

Gets the replay detection flag.

C++

```
mxt_result GetReplayDetection(OUT bool* pbReplayDetection);
```

Parameters

Parameters	Description
OUT bool* pbReplayDetection	Pointer to hold the replay detection flag.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the replay detection flag from the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.9 - CKerberosGssContextOptions::Release Method

Releases the Gcc context options object.

C++

```
void Release();
```

Description

Releases the current object.

NOTES: Delete MUST NOT be called on a CKerberosGssContextOptions (see page 441) object. The Release method is the proper way to do this.

2.9.1.6.2.10 - CKerberosGssContextOptions::SetAnonymousity Method

Sets the is anonymous flag.

C++

```
mxt_result SetAnonymousity(IN bool bAnonymous);
```

Parameters

Parameters	Description
IN bool bAnonymous	The Anonymous flag.

Returns

- resS_OK

Description

Sets the anonymous flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.11 - CKerberosGssContextOptions::SetConfidentiality Method

Sets the confidentiality flag.

C++

```
mxt_result SetConfidentiality(IN bool bConfidential);
```

Parameters

Parameters	Description
IN bool bConfidential	The confidentiality flag.

Returns

- resS_OK

Description

Sets the confidentiality flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.12 - CKerberosGssContextOptions::SetCredentialDelagation Method

Sets the credential delegation flag.

C++

```
mxt_result SetCredentialDelagation(IN bool bDelegateConfidential);
```

Parameters

Parameters	Description
IN bool bDelegateConfidential	The credential delegation flag.

Returns

- resS_OK

Description

Sets the credential delegation flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.13 - CKerberosGssContextOptions::SetIntegrity Method

Sets the integrity flag.

C++

```
mxt_result SetIntegrity(IN bool bIntegrity);
```

Parameters

Parameters	Description
IN bool bIntegrity	The integrity flag.

Returns

- resS_OK

Description

Sets the integrity flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.14 - CKerberosGssContextOptions::SetMutualAuthentication Method

Sets the mutual authentication flag.

C++

```
mxt_result SetMutualAuthentication(IN bool bMutualAuthentication);
```

Parameters

Parameters	Description
IN bool bMutualAuthentication	The mutual authentication flag.

Returns

- resS_OK

Description

Sets the mutual authentication flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.15 - CKerberosGssContextOptions::SetOutOfSequenceDetection Method

Sets the out of sequence detection flag.

C++

```
mxt_result SetOutOfSequenceDetection(IN bool bOutOfSequenceDetection);
```

Parameters

Parameters	Description
IN bool bOutOfSequenceDetection	The out of sequence detection flag.

Returns

- resS_OK

Description

Sets the out of sequence detection flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.6.2.16 - CKerberosGssContextOptions::SetReplayDetection Method

Sets the replay detection flag.

C++

```
mxt_result SetReplayDetection(IN bool bReplayDetection);
```

Parameters

Parameters	Description
IN bool bReplayDetection	The replay detection flag.

Returns

- resS_OK

Description

Sets the replay detection flag in the current CKerberosGssContextOptions (see page 441) object.

2.9.1.7 - CKerberosPrincipal Class

Class representing a Kerberos principal.

Class Hierarchy

```
CKerberosPrincipal
```

C++

```
class CKerberosPrincipal;
```

Description

This class represents a kerberos principal (e.g., a workstation user or a network server).

A kerberos principal is made up of two parts: the name and the realm. The name by itself may also be made up of multiple name components, separated by a slash. A full name is made up of a name, a @, and a realm.

namecomponent(/namecomponent)*@realm

Location

Kerberos/CKerberosPrincipal.h

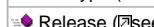
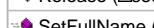
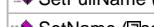
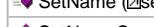
Destructors

Destructor	Description
 ~CKerberosPrincipal (see page 448)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 CloneFrom (see page 448)	Creates a copy of a CKerberosPrincipal object.
 GetFullName (see page 449)	Gets the full name of the principal.
 GetName (see page 449)	Gets the name of the principal.
 GetNameComponents (see page 449)	Gets the name components of the principal name.
 GetRealm (see page 450)	Gets the realm of the principal.
 GetType (see page 450)	Gets the type of the principal.
 Release (see page 450)	Releases the CKerberosPrincipal.
 SetFullName (see page 450)	Sets the full name of the principal.
 SetName (see page 451)	Sets the name of the principal.
 SetNameComponents (see page 451)	Sets the name components of the principal name.
 SetRealm (see page 451)	Sets the realm of the principal.
 SetType (see page 452)	Sets the type of the principal.

Legend

	Method
--	--------

2.9.1.7.1 - Constructors

2.9.1.7.1.1 - CKerberosPrincipal::~CKerberosPrincipal Destructor

Destructor.

C++

```
virtual ~CKerberosPrincipal();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a CKerberosPrincipal (see page 447) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.9.1.7.2 - Methods

2.9.1.7.2.1 - CKerberosPrincipal::CloneFrom Method

Creates a copy of a CKerberosPrincipal (see page 447) object.

C++

```
mxt_result CloneFrom(IN const CKerberosPrincipal* pPrincipal);
```

Parameters

Parameters	Description
IN const CKerberosPrincipal* pPrincipal	A pointer to a CKerberosPrincipal (see page 447) object.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resFE_MITOSFW_KERBEROS_INVALID_STATE
- resS_OK

Description

Creates a copy of a CKerberosPrincipal (see page 447) object.

2.9.1.7.2.2 - CKerberosPrincipal::GetFullName Method

Gets the full name of the principal.

C++

```
mxt_result GetFullName(OUT CString* pstrFullName);
```

Parameters

Parameters	Description
OUT CString* pstrFullName	A pointer to a CString (see page 126) to hold the full name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the full name from the CKerberosObject. A full name is returned in the following manner:
NameComponent(/NameComponent)@Realm

2.9.1.7.2.3 - CKerberosPrincipal::GetName Method

Gets the name of the principal.

C++

```
mxt_result GetName(OUT CString* pstrName);
```

Parameters

Parameters	Description
OUT CString* pstrName	A pointer to a CString (see page 126) to hold the name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the name from the CKerberosObject. A name is returned in the following manner: NameComponent(/NameComponent).

2.9.1.7.2.4 - CKerberosPrincipal::GetNameComponents Method

Gets the name components of the principal name.

C++

```
mxt_result GetNameComponents(OUT CVector<CString>* pvecstrComponents);
```

Parameters

Parameters	Description
OUT CVector<CString>* pvecstrComponents	A pointer to a Vector to hold the name components.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the names of the components from the CKerberosObject.

2.9.1.7.2.5 - CKerberosPrincipal::GetRealm Method

Gets the realm of the principal.

C++

```
mxt_result GetRealm(OUT CString* pstrRealm);
```

Parameters

Parameters	Description
OUT CString* pstrRealm	A pointer to a CString (see page 126) to hold the name of the realm.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the name of the realm from the CKerberosObject.

2.9.1.7.2.6 - CKerberosPrincipal::GetType Method

Gets the type of the principal.

C++

```
mxt_result GetType(OUT EType* peType);
```

Parameters

Parameters	Description
OUT EType* peType	A pointer to a EType (see page 464) to hold the principal type.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Gets the type of principal associated with the CKerberosObject.

2.9.1.7.2.7 - CKerberosPrincipal::Release Method

Releases the CKerberosPrincipal (see page 447).

C++

```
void Release();
```

Description

Releases the resources associated with the principal.

2.9.1.7.2.8 - CKerberosPrincipal::SetFullName Method

Sets the full name of the principal.

C++

```
mxt_result SetFullName(IN const CString* pstrFullName);
```

Parameters

Parameters	Description
IN const CString* pstrFullName	A pointer to a CString (see page 126) holding the full name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Sets the full name in the CKerberosObject. A full name is passed in in the following manner:
NameComponent(/NameComponent)@Realm

2.9.1.7.2.9 - CKerberosPrincipal::SetName Method

Sets the name of the principal.

C++

```
mxt_result SetName(IN const CString* pstrName);
```

Parameters

Parameters	Description
IN const CString* pstrName	A pointer to a CString (see page 126) holding the name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Sets the name in the CKerberosObject. A name is passed in in the following manner: NameComponent(/NameComponent).

2.9.1.7.2.10 - CKerberosPrincipal::SetNameComponents Method

Sets the name components of the principal name.

C++

```
mxt_result SetNameComponents(IN const CVector<CString>* pvecstrComponents);
```

Parameters

Parameters	Description
IN const CVector<CString>* pvecstrComponents	A pointer to a CVector (see page 227) holding the name components.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Sets the name components in the CKerberosObject.

2.9.1.7.2.11 - CKerberosPrincipal::SetRealm Method

Sets the realm of the principal.

C++

```
mxt_result SetRealm(IN const CString* pstrRealm);
```

Parameters

Parameters	Description
IN const CString* pstrRealm	A pointer to a CString (see page 126) holding the name of the realm.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Sets the realm in the CKerberosObject.

2.9.1.7.2.12 - CKerberosPrincipal::SetType Method

Sets the type of the principal.

C++

```
mxt_result SetType(IN EType eType);
```

Parameters

Parameters	Description
IN EType eType	ETpe containing the principal type.

Returns

- resS_OK

Description

Sets the principal type in the CKerberosObject.

2.9.1.8 - CKerberosRealm Class

A container that stores information about a realm.

Class Hierarchy

CKerberosRealm

C++

```
class CKerberosRealm;
```

Description

A container that stores information about a realm. Currently, the only supported information is the address of the KDC. A KDC (Key Distribution Center) is a server to which requests for tickets may be sent.

Location

Kerberos/CKerberosRealm.h

Constructors

Constructor	Description
CKerberosRealm (see page 453)	Default constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CKerberosRealm (see page 453)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
GetKeyDistributionCenterFqdn (see page 453)	Gets the fully qualified domain name of the key distribution center.
GetRealm (see page 454)	Gets the realm.
SetKeyDistributionCenterFqdn (see page 454)	Sets the fully qualified domain name of the key distribution center.
SetRealm (see page 454)	Sets the realm.

Legend

	Method
---	--------

2.9.1.8.1 - Constructors

2.9.1.8.1.1 - CKerberosRealm

2.9.1.8.1.1.1 - CKerberosRealm::CKerberosRealm Constructor

Default constructor.

C++

```
CKerberosRealm();
```

Description

Constructor.

2.9.1.8.1.1.2 - CKerberosRealm::CKerberosRealm Constructor

Copy constructor.

C++

```
CKerberosRealm(IN const CKerberosRealm& rRealm);
```

Parameters

Parameters	Description
IN const CKerberosRealm& rRealm	CKerberosRealm to copy.

Description

Copy constructor.

2.9.1.8.1.1.3 - CKerberosRealm::CKerberosRealm Constructor

Constructor.

C++

```
CKerberosRealm(IN const CString* pstrRealm, IN const CFqdn* pKeyDistributionCenter);
```

Parameters

Parameters	Description
IN const CString* pstrRealm	Pointer to a CString (See page 126) object holding the realm name.
IN const CFqdn* pKeyDistributionCenter	Pointer to a CFqdn (See page 527) holding the fully qualified domain name of the key distribution center.

Description

Constructor. Builds the Kerberos realm by using the provided realm name and fully qualified domain name.

2.9.1.8.2 - Destructors

2.9.1.8.2.1 - CKerberosRealm::~CKerberosRealm Destructor

Destructor.

C++

```
virtual ~CKerberosRealm();
```

Description

Destructor.

2.9.1.8.3 - Methods

2.9.1.8.3.1 - CKerberosRealm::GetKeyDistributionCenterFqdn Method

Gets the fully qualified domain name of the key distribution center.

C++

```
mxt_result GetKeyDistributionCenterFqdn(OUT CFqdn* pFqdn) const;
```

Parameters

Parameters	Description
OUT CFqdn* pFqdn	Pointer to hold the FQDN.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK;

Description

Gets the fully qualified domain name of the key distribution center associated with this CKerberosRealm (see page 452).

2.9.1.8.3.2 - CKerberosRealm::GetRealm Method

Gets the realm.

C++

```
mxt_result GetRealm(OUT CString* pstrRealm) const;
```

Parameters

Parameters	Description
OUT CString* pstrRealm	Pointer to hold the realm name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK;

Description

Gets name of the realm associated with this CKerberosRealm (see page 452) object.

2.9.1.8.3.3 - CKerberosRealm::SetKeyDistributionCenterFqdn Method

Sets the fully qualified domain name of the key distribution center.

C++

```
mxt_result SetKeyDistributionCenterFqdn(IN const CFqdn* pFqdn);
```

Parameters

Parameters	Description
IN const CFqdn* pFqdn	Pointer holding the FQDN.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK;

Description

Sets the fully qualified domain name of the key distribution center associated with this CKerberosRealm (see page 452).

2.9.1.8.3.4 - CKerberosRealm::SetRealm Method

Sets the realm.

C++

```
mxt_result SetRealm(IN const CString* pstrRealm);
```

Parameters

Parameters	Description
IN const CString* pstrRealm	Pointer holding the realm name.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK;

Description

Sets name of the realm associated with this CKerberosRealm (see page 452) object.

2.9.1.9 - CKerberosTicket Class

A container that stores information about a ticket.

Class Hierarchy

CKerberosTicket

C++

```
class CKerberosTicket;
```

Description

A container that stores information about a ticket.

Location

Kerberos/CKerberosTicket.h

Destructors

Destructor	Description
~CKerberosTicket (see page 455)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
GetClientPrincipal (see page 456)	Retrieves information about the client principal.
.GetServicePrincipal (see page 456)	Retrieves information about the service principal.
Release (see page 456)	Releases the ticket.

Legend

	Method
---	--------

2.9.1.9.1 - Destructors

2.9.1.9.1.1 - CKerberosTicket::~CKerberosTicket Destructor

Destructor.

C++

```
virtual ~CKerberosTicket();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a CKerberosTicket (see page 455) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.9.1.9.2 - Methods

2.9.1.9.2.1 - CKerberosTicket::GetClientPrincipal Method

Retrieves information about the client principal.

C++

```
mxt_result GetClientPrincipal(OUT CKerberosPrincipal* pClient) const;
```

Parameters

Parameters	Description
OUT CKerberosPrincipal* pClient	A pointer to a CKerberosPrincipal (see page 447) to hold the returned principal.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Retrieves the client principal from the current CKerberosTicket (see page 455).

2.9.1.9.2.2 - CKerberosTicket::GetServicePrincipal Method

Retrieves information about the service principal.

C++

```
mxt_result GetServicePrincipal(OUT CKerberosPrincipal* pService) const;
```

Parameters

Parameters	Description
OUT CKerberosPrincipal* pService	A pointer to a CKerberosPrincipal (see page 447) to hold the returned principal.

Returns

- resFE_MITOSFW_KERBEROS_INVALID_ARGUMENT
- resS_OK

Description

Retrieves the Service principal from the current CKerberosTicket (see page 455).

2.9.1.9.2.3 - CKerberosTicket::Release Method

Releases the ticket.

C++

```
void Release();
```

Description

Releases the resources associated with the ticket.

2.9.1.10 - CKerberosTicketOptions Class

Contains the state of all options to use when requesting a ticket.

Class Hierarchy

```
CKerberosTicketOptions
```

C++

```
class CKerberosTicketOptions;
```

Description

CKerberosTicketOptions contains the state of all the options to pass when requesting a new ticket or the renewal of an existing ticket.

Location

Kerberos/CKerberosTicketRequest.h

Destructors

Destructor	Description
~CKerberosTicketOptions (see page 457)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
CloneFrom (see page 457)	Copies the information from another ticket option.
GetAddresses (see page 458)	Gets the possible addresses.
GetForwardable (see page 458)	Gets the forwardable flag.
GetLifeTimeMs (see page 458)	Gets the possible lifetime in ms.
GetProxiable (see page 459)	Gets the proxiable flag.
GetRenewableLifeTimeMs (see page 459)	Gets the renewable lifetime in ms.
Release (see page 459)	Releases the ticket options.
SetAddresses (see page 459)	Sets the possible addresses.
SetForwardable (see page 460)	Only TGT
SetLifeTimeMs (see page 460)	Sets the possible lifetime in ms.
SetProxiable (see page 460)	Gets the proxiable flag.
SetRenewableLifeTimeMs (see page 460)	Gets the renewable lifetime in ms.

Legend

	Method
---	--------

2.9.1.10.1 - Destructors

2.9.1.10.1.1 - CKerberosTicketOptions::~CKerberosTicketOptions Destructor

Destructor.

C++

```
virtual ~CKerberosTicketOptions();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a CKerberosTicketOptions (see page 456) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.9.1.10.2 - Methods

2.9.1.10.2.1 - CKerberosTicketOptions::CloneFrom Method

Copies the information from another ticket option.

C++

```
mxt_result CloneFrom(IN const CKerberosTicketOptions* pOptions);
```

Parameters

Parameters	Description
IN const CKerberosTicketOptions* pOptions	A pointer to a CKerberosTicketOptions (see page 456) to copy.

Returns

- resFE_FAIL
- resS_OK

Description

Creates a copy of the CKerberosTicketOptions (see page 456) provided.

2.9.1.10.2.2 - CKerberosTicketOptions::GetAddresses Method

Gets the possible addresses.

C++

```
mxt_result GetAddresses(OUT CVector<CSocketAddr>* pvecAddresses) const;
```

Parameters

Parameters	Description
OUT CVector<CSocketAddr>* pvecAddresses	A pointer to a CVector (see page 227) to contain a list of addresses.

Returns

- resFE_FAIL
- resS_OK

Description

Gets the list of addresses that are issued in a ticket request.

2.9.1.10.2.3 - CKerberosTicketOptions::GetForwardable Method

Gets the forwardable flag.

C++

```
mxt_result GetForwardable(OUT bool* pbForwardable) const;
```

Parameters

Parameters	Description
OUT bool* pbForwardable	A pointer to hold the forwardable flag.

Returns

- resFE_FAIL
- resS_OK

Description

Gets the forwardable flag that is set in ticket requests.

2.9.1.10.2.4 - CKerberosTicketOptions::GetLifeTimeMs Method

Gets the possible lifetime in ms.

C++

```
mxt_result GetLifeTimeMs(OUT uint64_t* puLifeTimeMs) const;
```

Parameters

Parameters	Description
OUT uint64_t* puLifeTimeMs	A pointer to hold the lifetime in ms.

Returns

- resFE_FAIL
- resS_OK

Description

Gets the ticket lifetime that is set in ticket requests.

2.9.1.10.2.5 - CKerberosTicketOptions::GetProxiable Method

Gets the proxiable flag.

C++

```
mxt_result GetProxiable(OUT bool* pbProxiable) const;
```

Parameters

Parameters	Description
OUT bool* pbProxiable	A pointer to hold the proxiable flag.

Returns

- resFE_FAIL
- resS_OK

Description

Gets the proxiable flag that is set in ticket requests.

2.9.1.10.2.6 - CKerberosTicketOptions::GetRenewableLifeTimeMs Method

Gets the renewable lifetime in ms.

C++

```
mxt_result GetRenewableLifeTimeMs(OUT uint64_t* puRenewableLifeTimeMs) const;
```

Parameters

Parameters	Description
OUT uint64_t* puRenewableLifeTimeMs	A pointer to hold the renewable lifetime in ms.

Returns

- resFE_FAIL
- resS_OK

Description

Gets the renewable lifetime that is set in ticket requests.

2.9.1.10.2.7 - CKerberosTicketOptions::Release Method

Releases the ticket options.

C++

```
void Release();
```

2.9.1.10.2.8 - CKerberosTicketOptions::SetAddresses Method

Sets the possible addresses.

C++

```
mxt_result SetAddresses(IN const CVector<CSocketAddr>* pvecAddresses);
```

Parameters

Parameters	Description
IN const CVector<CSocketAddr>* pvecAddresses	A pointer to a CVector (see page 227) containing a list of addresses.

Returns

- resFE_FAIL
- resS_OK

Description

Sets the list of addresses that are issued in a ticket request.

2.9.1.10.2.9 - CKerberosTicketOptions::SetForwardable Method

Only TGT

C++

```
mxt_result SetForwardable(IN bool bForwardable);
```

Parameters

Parameters	Description
IN bool bForwardable	The forwardable flag.

Returns

- resS_OK

Description

Sets the forwardable flag that is set in ticket requests.

2.9.1.10.2.10 - CKerberosTicketOptions::SetLifeTimeMs Method

Sets the possible lifetime in ms.

C++

```
mxt_result SetLifeTimeMs(IN uint64_t uLifeTimeMs);
```

Parameters

Parameters	Description
IN uint64_t uLifeTimeMs	The lifetime in ms.

Returns

- resFE_FAIL
- resS_OK

Description

Sets the ticket lifetime that is set in ticket requests.

2.9.1.10.2.11 - CKerberosTicketOptions::SetProxiable Method

Gets the proxiable flag.

C++

```
mxt_result SetProxiable(IN bool bProxiable);
```

Parameters

Parameters	Description
IN bool bProxiable	The proxiable flag.

Returns

- resS_OK

Description

Sets the proxiable flag that is set in ticket requests.

2.9.1.10.2.12 - CKerberosTicketOptions::SetRenewableLifeTimeMs Method

Gets the renewable lifetime in ms.

C++

```
mxt_result SetRenewableLifeTimeMs(IN uint64_t uRenewableLifeTimeMs);
```

Parameters

Parameters	Description
IN uint64_t uRenewableLifeTimeMs	The renewable lifetime in ms.

Returns

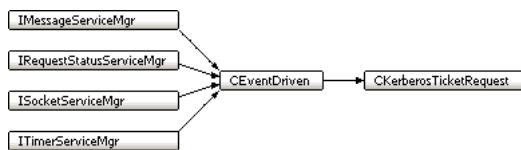
- resS_OK

Description

Sets the renewable lifetime that is set in ticket requests.

2.9.1.11 - CKerberosTicketRequest Class

Class implementing Kerberos ticket request.

Class Hierarchy**C++**

```
class CKerberosTicketRequest : protected CEventDriven;
```

Description

CKerberosTicketRequest

Location

Kerberos/CKerberosTicketRequest.h

Constructors**CEventDriven Class**

CEventDriven Class	Description
◆ CEventDriven (see page 764)	Constructor.

Legend

◆	Method
---	--------

Destructors**CEventDriven Class**

CEventDriven Class	Description
◆ ~CEventDriven (see page 764)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
◆ Release (see page 462)	Releases the object.

CEventDriven Class

CEventDriven Class	Description
◆ Activate (see page 764)	Associates a Servicing Thread with this Event Driven.
◆ DisableCompletionDetection (see page 765)	Disables the detection of request completion.
◆ DisableEventsDetection (see page 765)	Disables the detection of events.

• EnableCompletionDetection (see page 766)	Enables the detection of request completion. Enables the detection of request completion.
• EnableEventsDetection (see page 766)	Enables the detection of events.
• FinalizeAndReleaseA (see page 766)	Finalizes and releases an Event Driven.
• GetIEComUnknown (see page 766)	Returns a pointer to the Servicing Thread. AddRef is already called.
• GetServicingThread (see page 767)	Returns a pointer to the Servicing Thread. AddRef is already called.
• IsCurrentExecutionContext (see page 767)	Returns whether or not the code is executing within the current execution context.
• PostMessage (see page 767)	Pushes a new message into the message queue.
• RegisterRequestStatus (see page 768)	Registers a request status. Registers a request status.
• RegisterSocket (see page 768)	Registers a socket.
• Release (see page 768)	Releases an Event Driven.
• StartTimer (see page 769)	Starts a new linear timer.
• StopAllTimers (see page 770)	Stops all timers owned by a manager.
• StopTimer (see page 770)	Stops a timer owned by a manager.
• UnregisterRequestStatus (see page 770)	Unregisters a request status. Unregisters a request status.
• UnregisterSocket (see page 771)	Unregisters a socket.

ITimerServiceMgr Class

ITimerServiceMgr Class	Description
• A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

ISocketServiceMgr Class

ISocketServiceMgr Class	Description
• A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

IRequestStatusServiceMgr Class

IRequestStatusServiceMgr Class	Description
• A EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
A	abstract

2.9.1.11.1 - Methods

2.9.1.11.1.1 - CKerberosTicketRequest::Release Method

Releases the object.

C++

```
void Release();
```

Description

Releases the resources associated with the ticket request.

2.9.1.12 - IKerberosTicketRequestMgr Class

This is the interface through which the kerberos ticket request reports its events.

Class Hierarchy

IKerberosTicketRequestMgr

C++

```
class IKerberosTicketRequestMgr;
```

Description

This is the interface through which the kerberos ticket request reports its events.

Location

Kerberos/IKerberosTicketRequestMgr.h

See Also

CKerberos (see page 426), CKerberosTicket (see page 455), CKerberosTicketRequest (see page 461)

Methods

Method	Description
◆ A EvKerberosTicketRequestMgrGetTicketError (see page 463)	Notifies that the get ticket request failed.
◆ A EvKerberosTicketRequestMgrGetTicketSuccess (see page 463)	Notifies that the get ticket request succeeded.

Legend

◆	Method
A	abstract

2.9.1.12.1 - Methods

2.9.1.12.1.1 - IKerberosTicketRequestMgr::EvKerberosTicketRequestMgrGetTicketError Method

Notifies that the get ticket request failed.

C++

```
virtual void EvKerberosTicketRequestMgrGetTicketError(IN mxt_opaque opq, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the request.
IN mxt_result res	The reported error.

Description

This is the event generated by an asynchronous get ticket request when it fails.

2.9.1.12.1.2 - IKerberosTicketRequestMgr::EvKerberosTicketRequestMgrGetTicketSuccess Method

Notifies that the get ticket request succeeded.

C++

```
virtual void EvKerberosTicketRequestMgrGetTicketSuccess(IN mxt_opaque opq, IN GO CKerberosTicket* pTicket) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the request.
IN GO CKerberosTicket* pTicket	A pointer to the requested ticket.

Description

This is the event generated by an asynchronous get ticket request when it succeeds.

2.9.2 - Enumerations

This section documents the enumerations of the Sources/Kerberos folder.

Enumerations

Enumeration	Description
EType (see page 464)	Types of Kerberos principal names.

2.9.2.1 - CKerberosPrincipal::EType Enumeration

Types of Kerberos principal names.

C++

```
enum EType {
    eTYPE_PRINCIPAL,
    eTYPE_SRV_INST,
    eTYPE_SRV_HST,
    eTYPE_UNKNOWN
};
```

Description

Holds the types of principal names. This indicates what kind of information is implied by the name.

Members

Members	Description
eTYPE_PRINCIPAL	Just the name of the principal.
eTYPE_SRV_INST	Service and other unique instance.
eTYPE_SRV_HST	Service with host name as instance.
eTYPE_UNKNOWN	Name type not known

2.10 - Kernel

This section documents the Sources/Kernel folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 464)
- Enumerations (see page 504)
- Functions (see page 506)
- Macros (see page 509)
- Structures (see page 511)
- Templates (see page 515)
- Types (see page 521)
- Variables (see page 521)

2.10.1 - Classes

This section documents the classes of the Sources/Kernel folder.

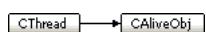
Classes

Class	Description
CAliveObj (see page 464)	Manages the life of an object.
CBinarySemaphore (see page 467)	Synchronization object with two states.
CCountingSemaphore (see page 470)	Synchronization object with a lock count.
CFile (see page 472)	Class handling file manipulation.
CFileTools (see page 478)	Tools for file management.
CMemoryAllocator (see page 480)	Class used to override new and delete default operators.
CMemoryBlock (see page 485)	Class used to store an allocated memory block.
IMemoryBlockAccumulator (see page 488)	Interface defining an accumulator of CMemoryBlocks.
CMutex (see page 488)	Basic synchronization object.
COsVersion (see page 490)	This class gives access to operating system version.
CThread (see page 491)	Allows the creation of a new thread of execution.
CTimer (see page 501)	Class implementing timers.

2.10.1.1 - CAliveObj Class

Manages the life of an object.

Class Hierarchy



C++

```
class CAliveObj : public CThread;
```

Description

The responsibility of the Alive Object is to manage the life of an object. Each Alive Object is associated to one thread. Part of its responsibilities are to start the thread and to manage the internal life state of the thread. The Alive Object is an abstract base class, therefore it is designed to be specialized by other classes. At its creation the Alive Object is in an unborn state (the associated thread has never been started). The Alive Object becomes alive when its member function Activate (see page 466) is called, at which point the associated thread is started. Following the termination of the thread, the Alive Object becomes dead. It is then possible to call Activate (see page 466) again.

On every known Operating System, when a thread is started it needs a pointer to a function. This function is the equivalent of the main for this thread. In the context of the Alive Object this main function is called the "Behavior" of the Alive Object. The Behavior member function is a pure virtual function, therefore it must be defined in a class derived from Alive Object.

The normal way to end a thread is to exit from its main function (Behavior member function), therefore the Terminate member function must somehow indicate to the Behavior member function to end.

The Alive Object termination is not managed by CAliveObj. This is mainly because there are many ways to accomplish this. Upon thread termination, the method BehaviorTerminating is called. This allows the inheriting class to signal its termination with the mechanism of its choice.

Constructors

Constructor	Description
• CAliveObj (see page 466)	Constructor.

CThread Class

CThread Class	Description
• CThread (see page 493)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CAliveObj (see page 466)	Destructor.

CThread Class

CThread Class	Description
• ~CThread (see page 493)	Destructor.

Legend

•	Method
V	virtual

Operators**CThread Class**

CThread Class	Description
• != (see page 500)	Different than operator.
• == (see page 500)	Comparison operator.

Legend

•	Method
---	--------

Methods

Method	Description
• Activate (see page 466)	Gives life to the object (Start the associated thread).
• IsAlive (see page 467)	Indicates the state of the object (thread-safe).
• IsDead (see page 467)	Indicates the state of the object (thread-safe).
• IsUnborn (see page 467)	Indicates the state of the object (thread-safe).

CThread Class

CThread Class	Description
• CreateKey (see page 493)	Creates a new TSD key.
• DeleteKey (see page 494)	Deletes the specified TSD key.
• FinalizeTsd (see page 494)	Finalizes the TSD.
• GetCurrentId (see page 494)	Gets the application ID of the current thread of execution.
• GetId (see page 495)	Gets the application ID of this thread.
• GetName (see page 495)	Gets the name of the thread.
• GetNativeThreadId (see page 495)	Gets the native thread ID.
• GetSpecific (see page 495)	Gets the data for the specified key.
• GetState (see page 496)	Gets the current state of the thread.
• GetThreadSelf (see page 496)	Gets the address of the current thread.
• GetThreadStackInfo (see page 497)	Gets the stack information of a specific CThread (see page 491).
• InitializeTsd (see page 498)	Initializes the TSD.
• IsCurrentThread (see page 498)	Checks if thread is current thread of execution.
• Join (see page 498)	Waits for thread termination.
• ResetThreadStackInfo (see page 499)	Resets the stack information for the current CThread (see page 491).
• SetSpecific (see page 499)	Sets new data for specified key.
• StartThread (see page 499)	Starts a new thread.

Legend

•	Method
---	--------

2.10.1.1.1 - Constructors

2.10.1.1.1.1 - CAliveObj::CAliveObj Constructor

Constructor.

C++

```
CAliveObj();
```

Description

Constructor.

Until the first activation, the object is "Unborn". After activation, the object is "Alive". When the thread is terminated, the object is "Dead".

The Semaphore used to wait for the thread creation MUST be start blocked (this is the default behavior of a Binary Semaphore).

2.10.1.1.2 - Destructors

2.10.1.1.2.1 - CAliveObj::~CAliveObj Destructor

Destructor.

C++

```
virtual ~CAliveObj();
```

Description

Destructor

Warning

It is wrong to destroy an object that is still alive! It must be terminated first.

2.10.1.1.3 - Methods

2.10.1.1.3.1 - CAliveObj::Activate Method

Gives life to the object (Start the associated thread).

C++

```
mxt_result Activate(IN const char* pszName = NULL, IN uint32_t uStackSize = 0, IN CThread::EPriority ePriority = CThread::eNORMAL);
```

Parameters

Parameters	Description
IN const char* pszName = NULL	The name of the associated thread. On some operating systems, it will be used as a system wide identifier for the thread. Does not have to be unique.
IN uint32_t uStackSize = 0	The stack size of the associated thread. 0 as the default stack size is ok.
IN CThread::EPriority ePriority = CThread::eNORMAL	The priority of the associated thread.

Returns

True if the object is "Alive", false otherwise.

Description

At its creation the Alive Object is "Unborn" (the associated thread is not started yet). The Alive Object becomes "Alive" when its member method Activate() is called. At this point the associated thread is started.

2.10.1.1.3.2 - CAliveObj::IsAlive Method

Indicates the state of the object (thread-safe).

C++

```
bool IsAlive();
```

2.10.1.1.3.3 - CAliveObj::IsDead Method

Indicates the state of the object (thread-safe).

C++

```
bool IsDead();
```

2.10.1.1.3.4 - CAliveObj::IsUnborn Method

Indicates the state of the object (thread-safe).

C++

```
bool IsUnborn();
```

2.10.1.2 - CBinarySemaphore Class

Synchronization object with two states.

Class Hierarchy

```
CBinarySemaphore
```

C++

```
class CBinarySemaphore;
```

Description

The CBinarySemaphore object is a synchronization object with two states: Acquired and Free. Once a binary semaphore is acquired, any following call to the Wait (see page 469) method will block until the semaphore is released using the Signal (see page 469) method, the given timeout has expired (for waitable semaphores only) or the semaphore is destroyed. CBinarySemaphores are not re-entrant.

A CBinarySemaphore can be created with or without the timed wait option. Semaphores that do not have the timed wait option are at least as fast as the ones with the option set but may be faster depending on the implementation. Therefore, it is recommended to set the timed wait option to false when timed wait is not required.

By default, the CBinarySemaphore is created acquired, with the timed wait option set.

Location

Kernel/CSemaphore.h

See Also

CMutex (see page 488), CCountingSemaphore (see page 470)

Constructors

Constructor	Description
CBinarySemaphore (see page 468)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CBinarySemaphore (see page 468)	Destructor.

Legend

	Method
--	--------

Methods

Method	Description
GetHandle (see page 469)	Gets the handle of the semaphore.
Signal (see page 469)	Signals a semaphore.
Wait (see page 469)	Acquires the semaphore.

Legend

	Method
--	--------

2.10.1.2.1 - Constructors**2.10.1.2.1.1 - CBinarySemaphore::CBinarySemaphore Constructor**

Constructor.

C++

```
CBinarySemaphore(IN EInitValue eVal = eACQUIRED, IN bool bAllowTimedWait = true);
```

Parameters

Parameters	Description
IN EInitValue eVal = eACQUIRED	Initial value of the semaphore. Can be one of eACQUIRED (default) or eFREE.
IN bool bAllowTimedWait = true	Sets the timed wait option, making it possible or not to wait on a semaphore up to the specified timeout. Otherwise, only infinite wait or no wait are permitted.

Description

Constructs a CBinarySemaphore synchronization object.

2.10.1.2.2 - Destructors**2.10.1.2.2.1 - CBinarySemaphore::~CBinarySemaphore Destructor**

Destructor.

C++

```
~CBinarySemaphore();
```

Description

Destructor.

2.10.1.2.3 - Methods

2.10.1.2.3.1 - CBinarySemaphore::GetHandle Method

Gets the handle of the semaphore.

C++

```
HANDLE GetHandle();
```

Returns

The handle to this object.

Description

Gets the Windows handle of the semaphore.

Notes

This method is for Win32 and WinCE implementations only.

2.10.1.2.3.2 - CBinarySemaphore::Signal Method

Signals a semaphore.

C++

```
void Signal(IN bool bReschedule = false) const;
```

Parameters

Parameters	Description
IN bool bReschedule = false	Once the semaphore is signalled, ask for rescheduling. There is no guaranty that the current thread will release control.

Returns

Nothing.

Description

Signals a semaphore, making available to another (possibly pending) thread.

2.10.1.2.3.3 - Wait

2.10.1.2.3.3.1 - CBinarySemaphore::Wait Method

Acquires the semaphore.

C++

```
bool Wait() const;
```

Returns

True if the semaphore has been acquired. False if there was an error.

Description

Tries to acquire the semaphore with regards to the current count.

2.10.1.2.3.3.2 - CBinarySemaphore::Wait Method

Waits on a semaphore.

C++

```
bool Wait(IN uint64_t uWaitTimeMs) const;
```

Parameters

Parameters	Description
IN uint64_t uWaitTimeMs	When the timed wait option is set, the maximum time in ms to wait before failing to acquire the semaphore. Another possible value, also valid when the timed wait option is not set, is 0 (no wait or "try wait").

Returns

True if the semaphore has been acquired. False if there was an error, if the timeout expired (timed wait option set) or if a timeout was given to Wait on a non time waitable semaphore.

Description

Tries to acquire the semaphore with regards to the uWaitTimeMs parameter, the timed wait option and the current count.

2.10.1.3 - CCountingSemaphore Class

Synchronization object with a lock count.

Class Hierarchy

 CCountingSemaphore

C++

```
class CCountingSemaphore;
```

Description

The Semaphore object is a synchronization object. The Semaphore object holds a lock count, which represents the number of times that any threads can Wait (see page 472) to acquire the Semaphore. Once the count of the semaphore has reached zero, any thread that waits to acquire it will have to wait until another thread releases the semaphore with the Signal (see page 471) method (or until the time-out value expires or until the semaphore is deleted).

A CCountingSemaphore can be created with or without the timed wait option. Semaphores that do not have the timed wait option are at least as fast as the ones with the option set but may be faster depending on the implementation. Therefore, it is recommended to set the timed wait option to false when timed wait is not required.

By default, the CCountingSemaphore is created all acquired, with the timed wait option set.

Location

Kernel/CSemaphore.h

See Also

CMutex (see page 488), CBinarySemaphore (see page 467)

Constructors

Constructor	Description
 CCountingSemaphore (see page 471)	Constructor.

Legend

 Method

Destructors

Destructor	Description
 ~CCountingSemaphore (see page 471)	Destructor.

Legend

 Method

Methods

Method	Description
 GetHandle (see page 471)	Gets the handle of the semaphore.
 Signal (see page 471)	Signals a semaphore.
 Wait (see page 472)	Acquires the semaphore.

Legend

 Method

2.10.1.3.1 - Constructors

2.10.1.3.1.1 - CCountingSemaphore::CCountingSemaphore Constructor

Constructor.

C++

```
CCountingSemaphore(IN uint32_t uStartCount = static_cast<uint32_t>(eALL_ACQUIRED), IN bool bAllowTimedWait = true);
```

Parameters

Parameters	Description
IN uint32_t uStartCount = static_cast<uint32_t>(eALL_ACQUIRED)	Initial value of the semaphore count. Can be any value between default value 0 (eALL_ACQUIRED) and uSEM_MAX_COUNT (eALL_FREE).
IN bool bAllowTimedWait = true	Sets the timed wait option, making it possible or not to wait on a semaphore up to the specified timeout. Otherwise, only infinite wait or no wait are permitted.

Description

Constructs a CCountingSemaphore synchronization object.

2.10.1.3.2 - Destructors

2.10.1.3.2.1 - CCountingSemaphore::~CCountingSemaphore Destructor

Destructor.

C++

```
~CCountingSemaphore();
```

Description

Destructor.

2.10.1.3.3 - Methods

2.10.1.3.3.1 - CCountingSemaphore::GetHandle Method

Gets the handle of the semaphore.

C++

```
HANDLE GetHandle();
```

Returns

The handle to this object.

Description

Gets the Windows handle of the semaphore.

Notes

This method is for Win32 and WinCE implementations only.

2.10.1.3.3.2 - CCountingSemaphore::Signal Method

Signals a semaphore.

C++

```
void Signal(IN bool bReschedule = false) const;
```

Parameters

Parameters	Description
IN bool bReschedule = false	Once the semaphore is signalled, ask for rescheduling. There is no guaranty that the current thread will release control.

Returns

Nothing.

Description

Signals a semaphore, incrementing the current count.

2.10.1.3.3.3 - Wait**2.10.1.3.3.3.1 - CCountingSemaphore::Wait Method**

Acquires the semaphore.

C++

```
bool Wait() const;
```

Returns

True if the semaphore has been acquired. False if there was an error.

Description

Tries to acquire the semaphore with regards to the current count.

2.10.1.3.3.3.2 - CCountingSemaphore::Wait Method

Waits on a semaphore.

C++

```
bool Wait(IN uint64_t uWaitTimeMs) const;
```

Parameters

Parameters	Description
IN uint64_t uWaitTimeMs	When the timed wait option is set, the maximum time in ms to wait before failing to acquire the semaphore. Another possible value, also valid when the timed wait option is not set, is 0 (no wait or "try wait").

Returns

True if the semaphore has been acquired. False if there was an error, if the timeout expired (timed wait option set) or if a timeout was given to Wait on a non time waitable semaphore.

Description

Tries to acquire the semaphore with regards to the current count, the uWaitTimeMs parameter, the timed wait option and the current count.

2.10.1.4 - CFile Class

Class handling file manipulation.

Class Hierarchy

```
CFile
```

C++

```
class CFile;
```

Description

CFile is the class that handles file manipulation. It provides a common interface with unbuffered methods for file manipulation on all operating systems.

It is essentially a wrapper above the operating system file's primitives. It does not support operations on directories. Thus it does not support creating a file in a new directory using the open method.

CFile can be safely shared between multiple threads. This means that all operations from each thread will be executed but the application is responsible to synchronize these operations from each thread.

Location

Kernel/CFile.h

Constructors

Constructor	Description
 CFile (see page 473)	Default Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CFile (see page 473)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 Close (see page 474)	Closes the file. Returns resS_OK if the operation succeeded.
 GetFileDescriptor (see page 474)	Gets the file descriptor.
 Open (see page 474)	Opens a file in the file system.
 Read (see page 475)	Reads the entire file into a blob.
 Seek (see page 476)	Changes the value of the offset of the file descriptor.
 Stat (see page 476)	Gets the file information.
 Truncate (see page 477)	Truncates a file.
 Write (see page 477)	Writes the entire blob into the file.

Legend

	Method
---	--------

2.10.1.4.1 - Constructors**2.10.1.4.1.1 - CFile::CFile Constructor**

Default Constructor.

C++`CFile();`**Description**

Constructor.

2.10.1.4.2 - Destructors**2.10.1.4.2.1 - CFile::~CFile Destructor**

Destructor.

C++`virtual ~CFile();`**Description**

Destructor.

2.10.1.4.3 - Methods

2.10.1.4.3.1 - CFile::Close Method

Closes the file. Returns resS_OK if the operation succeeded.

C++

```
mxt_result Close();
```

Returns

- resS_OK: File closed successfully or it was already closed.
- resFE_FAIL: Cannot close file.

Description

Closes the file. The file descriptor will be invalid after this.

2.10.1.4.3.2 - CFile::GetFileDescriptor Method

Gets the file descriptor.

C++

```
mxt_fd GetFileDescriptor();
```

Returns

The file descriptor or a pointer to the RFile object. Returns -1 for an invalid file descriptor.

Description

Gets the file descriptor of the currently opened file.

The file descriptor returned should not be kept as it can be invalidated by the following methods:

- Truncate (see page 477)
- Close (see page 474)
- Stat (see page 476) // Windows CE

2.10.1.4.3.3 - CFile::Open Method

Opens a file in the file system.

C++

```
mxt_result Open(IN const char* pszPathName, IN EFlags eFlags, IN EMode eMode);
```

Parameters

Parameters	Description
IN const char* pszPathName	Path and name of the file to open.
IN EFlags eFlags	Flags to use to open the file.
IN EMode eMode	Mode to set if the file is created.

Returns

- resS_OK: File opened successfully.
- resFE_INVALID_ARGUMENT: Cannot open file. Pathname may be null or the flag combination is not allowed.
- resFE_INVALID_STATE: Cannot open file. This CFile (see page 472) is already in use.
- resFE_FAIL: Cannot open file. It can be already opened, it does not exist or you may not have permission to open it.

Description

Opens a file in the file system.

Only one of these flags can be used at a time:

- eREAD_ONLY
- eWRITE_ONLY
- eREAD_WRITE.

These flags cannot be mixed. Also the flags eCREATE and eTRUNCATE cannot be used with the eREAD_ONLY flag.

The eREAD_WRITE and eBINARY flags are the default values when no flag is specified.

Warning

Opening the same file twice or more is not supported and it may lead to unpredictable behaviour.

2.10.1.4.3.4 - Read

2.10.1.4.3.4.1 - CFile::Read Method

Reads the entire file into a blob.

C++

```
mxt_result Read(OUT CBlob* pBlob);
```

Parameters

Parameters	Description
OUT CBlob* pBlob	Blob to hold the bytes read.

Returns

- resS_OK: Entire file read successfully.
- resFE_INVALID_ARGUMENT: Cannot read file. pBlob is NULL.
- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot read bytes. Read access may be denied.

Description

Reads the entire file into a blob. Read starts at the beginning of the file. When the blob is not empty, its content is overwritten.

The file pointer will be at the end of the file after this operation.

2.10.1.4.3.4.2 - CFile::Read Method

Reads bytes from the file into a blob.

C++

```
mxt_result Read(OUT CBlob* pBlob, IN unsigned int uSizeInBytes);
```

Parameters

Parameters	Description
OUT CBlob* pBlob	Blob to hold the bytes read.
IN unsigned int uSizeInBytes	Number of bytes to read.

Returns

- resS_OK: Bytes read successfully.
- resFE_INVALID_ARGUMENT: Cannot read bytes. pBlob is NULL.
- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot read bytes. Read access may be denied.

Description

Reads uSizeInBytes bytes from the file into a blob. Read starts at current position in the file. When the blob is not empty, its content is overwritten. The new blob size can be less than uSizeInBytes because it is the number of bytes read from the file.

2.10.1.4.3.4.3 - CFile::Read Method

Reads bytes from the file.

C++

```
int Read(OUT void* uBuffer, IN unsigned int uSizeInBytes);
```

Parameters

Parameters	Description
IN unsigned int uSizeInBytes	Number of bytes to read.
pBuffer	Buffer to hold the number of bytes read.

Returns

The number of bytes read or -1 when pBuffer is null or there is an error. It also returns -1 when no file opened.

Description

Reads up to uSizeInBytes bytes from the file to the read buffer. This method can read less than uSizeInBytes if the end of file is reached.

For example, reading from a file already at the end of file returns 0 bytes.

To be able to read in a file, the eREAD_ONLY or eREAD_WRITE flags must be used.

Notes

It is up to the application to make sure the buffer is large enough to contain uSizeInBytes + 1.

2.10.1.4.3.5 - CFile::Seek Method

Changes the value of the offset of the file descriptor.

C++

```
int Seek(IN int nOffsetInBytes, IN ESeekPositions eSeekPosition);
```

Parameters

Parameters	Description
IN int nOffsetInBytes	The new offset in the file.
IN ESeekPositions eSeekPosition	Position from which to apply the offset.

Returns

The resulting offset from the beginning of the file or -1 on an error. It also returns -1 when no file opened.

Description

Changes the value of the offset where the next read or write operation will begin.

Seeking before the beginning of the file is not allowed. In this case, the offset is kept unchanged and an error is returned.

Seeking beyond end of file is accepted but file extention is done only on the next write operation. In this case, the offset is changed to the desired offset.

2.10.1.4.3.6 - CFile::Stat Method

Gets the file information.

C++

```
mxt_result Stat(OUT struct SStat* pstStat);
```

Parameters

Parameters	Description
OUT struct SStat* pstStat	Structure to hold the file information.

Returns

- resS_OK: File's stats retrieved successfully.
- resFE_INVALID_ARGUMENT: Cannot retrieve file's stats. pstStat is NULL.
- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot retrieve file's stats. File may not exist.

Description

Fills the stat structure with the file information. Not all the fields of the struture are filled depending on the operating system:

- On Windows CE, only these fields are retrieved: uMode, nSize, nModificationTime, nAccessTime, nChangeTime and uAttribute.
- On Symbian, only these fields are retrieved: nSize, nModificationTime and uAttribute.
- On the other operating systems, these fields are retrieved plus the following in the last two points of the list depending on the operating system:

uDevice, uINode, uMode, uLinkCount, nUserId, nGroupId, uDeviceType, nSize, nAccessTime, nModificationTime and nChangeTime.

- The following are also supported on Linux, Solaris, VxWorks:

nBlockSize and nBlockCount

- VxWorks also supports:

uAttribute

Warning

This method invalidates the file descriptor on the Windows CE operating system.

2.10.1.4.3.7 - CFile::Truncate Method

Truncates a file.

C++

```
mxt_result Truncate(IN int nOffsetInBytes);
```

Parameters

Parameters	Description
IN int nOffsetInBytes	The offset at which to truncate or extend the file.

Returns

- resS_OK: File truncated or extended successfully.
- resFE_INVALID_ARGUMENT: nOffsetInBytes cannot be negative.
- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot truncate or extend file. It can be already opened or it does not exist.

Description

Truncates a file at the specified offset or extends a file to a specified offset.

The current position in the file remains unchanged unless the current file offset is not invalid according to the new file size. In that case, the current position is set to the new end of file.

Warning

This method invalidates the file descriptor on Nucleus, VxWorks and Windows CE operating systems.

2.10.1.4.3.8 - Write

2.10.1.4.3.8.1 - CFile::Write Method

Writes the entire blob into the file.

C++

```
mxt_result Write(IN const CBlob& rBlob);
```

Parameters

Parameters	Description
IN const CBlob& rBlob	Buffer to write to the file.

Returns

- resS_OK: Bytes written successfully.

- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot write bytes. Write access may be denied.

Description

Writes the entire blob into the file. Write starts at the current position in the file.

2.10.1.4.3.8.2 - CFile::Write Method

Writes bytes from a blob into the file.

C++

```
mxt_result Write(IN const CBlob& rBlob, IN unsigned int uSizeInBytes);
```

Parameters

Parameters	Description
IN const CBlob& rBlob	Blob to write to the file.
IN unsigned int uSizeInBytes	Number of bytes to write.

Returns

- resS_OK: Bytes written successfully.
- resFE_INVALID_STATE: No file opened.
- resFE_FAIL: Cannot write bytes. Write access may be denied.

Description

Writes either the specified size or the CBlob (see page 95) size, whichever is smallest. Write starts at the current position in the file.

2.10.1.4.3.8.3 - CFile::Write Method

Writes the buffer in the file.

C++

```
int Write(IN const void* pBuffer, IN unsigned int uSizeInBytes);
```

Parameters

Parameters	Description
IN unsigned int uSizeInBytes	Number of bytes to write.
pBuffer	Buffer to write to the file.

Returns

The actual number of bytes written or -1 when pBuffer is null or an error occurred. It also returns -1 when no file opened.

Description

Writes uSizeInBytes bytes from the buffer to the file.

The bytes are written at the current file position unless the eAPPEND flag was specified on Open (see page 474). If eAPPEND is specified, all bytes are written to the end of the file.

2.10.1.5 - CFileTools Class

Tools for file management.

Class Hierarchy

```
[CFileTools]
```

C++

```
class CFileTools;
```

Description

CFileTools is the class that provides a common interface for file management on all operating systems. The file must not be opened when using these methods.

Location

Kernel/CFileTools.h

Methods

Method	Description
ChangeMode (see page 479)	Changes the mode of a file.
CreateNewDir (see page 479)	Creates a directory.
Remove (see page 480)	Removes a file or a directory.
Rename (see page 480)	Renames a file. It will also move it to another directory if the path has been changed.

Legend

	Method
--	--------

2.10.1.5.1 - Methods

2.10.1.5.1.1 - CFileTools::ChangeMode Method

Changes the mode of a file.

C++

```
static mxt_result ChangeMode(IN const char* pszPathName, IN Cfile::EMode eMode);
```

Parameters

Parameters	Description
IN const char* pszPathName	Path and name of the file for which to modify attributes.
IN Cfile::EMode eMode	New file mode to set.

Returns

- resS_OK if the file attributes were modified, resFE_FAIL otherwise.
- resFE_NOT_IMPLEMENTED: the method is not implemented on this operating system.

Description

Modifies a file's attributes.

This function behaves differently on different operating systems. Please refer to the SetFileAttributes method documentation for the Windows operating system and the chmod method documentation for other operating systems.

Both Vxworks and Nucleus do not support changing file modes.

2.10.1.5.1.2 - CFileTools::CreateNewDir Method

Creates a directory.

C++

```
static mxt_result CreateNewDir(IN const char* pszPathName);
```

Parameters

Parameters	Description
IN const char* pszPathName	Path and name of the directory to create.

Returns

- resS_OK: the directory was created.
- resFE_FAIL: the directory was not created.
- resFE_NOT_IMPLEMENTED: Operating system does not support creating directories.

Description

Creates a directory.

Nucleus do not support creating directories. On VxWorks (versions 5.4, 5.5 and 6.5) the file system must be properly configured to be able to add directories.

2.10.1.5.1.3 - CFileTools::Remove Method

Removes a file or a directory.

C++

```
static mxt_result Remove(IN const char* pszPathName);
```

Parameters

Parameters	Description
IN const char* pszPathName	Path and name of the file or directory to remove.

Returns

resS_OK if the file or directory was removed, resFE_FAIL otherwise.

Description

Removes a file or a directory from the file system. The file must be closed before attempting this operation.

Notes

On the Linux, Solaris and VxWorks operating systems, it is allowed to remove a file that is opened. On all other operating systems, an error occurs when trying to remove an opened file.

2.10.1.5.1.4 - CFileTools::Rename Method

Renames a file. It will also move it to another directory if the path has been changed.

C++

```
static mxt_result Rename(IN const char* pszOldPathName, IN const char* pszNewPathName);
```

Parameters

Parameters	Description
IN const char* pszOldPathName	Old path and name of the file to rename.
IN const char* pszNewPathName	New path and name of the file to rename. This file must not exist.

Returns

resS_OK if the file was renamed, resFE_FAIL otherwise.

Description

Renames a file in the file system. Rename fails when the file pszNewPathName already exists or when the source file does not exist. The file must be closed before using this method.

Notes

On the Linux, Solaris and VxWorks operating systems, it is allowed to rename a file that is opened. On all other operating systems, an error occurs when trying to rename an opened file.

2.10.1.6 - CMemoryAllocator Class

Class used to override new and delete default operators.

Class Hierarchy

```
CMemoryAllocator
```

C++

```
class CMemoryAllocator;
```

Description

```
class CMemoryAllocator
```

This class is used to override the new and delete default operators. It is also used to access memory usage information.

Methods

Method	Description
 Allocate (see page 481)	Allocates a block of memory.

Deallocation (see page 482)	Frees a block of memory.
DisableMemoryStatistics (see page 482)	Disables memory statistics acquisition.
EnableMemoryStatistics (see page 482)	Enables memory statistics acquisition.
EnumMemoryBlocks (see page 483)	Enumerates through the currently allocated list of memory blocks.
GetMemoryBlock (see page 483)	Get a pointer to the CMemoryBlock (see page 485) associated with an allocated memory block.
GetMemoryStatistics (see page 484)	Gets a structure containing current memory statistics.
ResetMemoryStatistics (see page 484)	Resets the structure containing current memory statistics.
SetMemoryStatistics (see page 484)	Sets the structure containing current memory statistics.

Legend

	Method
--	--------

2.10.1.6.1 - Methods

2.10.1.6.1.1 - Allocate

2.10.1.6.1.1.1 - CMemoryAllocator::Allocate Method

Allocates a block of memory.

C++

```
static void* GO_Allocate(IN size_t uSize);
```

Parameters

Parameters	Description
IN size_t uSize	Size of the memory block to allocate.

Returns

A void pointer to the allocated memory block.

Description

This method is used to allocate a block of memory that may be deallocated by passing a pointer located at any valid arbitrary location inside the allocated block. Normally, the allocated block must be deallocated by passing the exact same pointer to Deallocation (see page 482).

See Also

Deallocation (see page 482)

2.10.1.6.1.1.2 - CMemoryAllocator::Allocate Method

Allocates a block of memory and stores extra information about the allocation.

C++

```
static void* GO_Allocate(IN size_t uSize, IN SMemoryBlockExtraInfo const& rExtraInfo);
```

Parameters

Parameters	Description
IN size_t uSize	Size (in bytes) of the memory allocation request.
IN SMemoryBlockExtraInfo const& rExtraInfo	Reference to a structure containing extra information about the memory allocation.

Returns

A void pointer to the user-accessible allocated memory.

Description

This method allocates a block of memory similarly to the CMemoryAllocator::Allocation method taking only one parameter. However, in addition of allocating the memory, this method takes an extra parameter used to give the CMemoryAllocator (see page 480) some extra information about the memory allocation which will be stored and can be accessed later through the EnumMemoryBlock or GetMemoryBlock (see page 483) methods.

See Also

Allocate

2.10.1.6.1.2 - Deallocate**2.10.1.6.1.2.1 - CMemoryAllocator::Deallocate Method**

Frees a block of memory.

C++

```
static void Deallocate(IN TOA const void* pvoidBlock);
```

Parameters

Parameters	Description
IN TOA const void* pvoidBlock	A pointer to the memory block to free.

Description

This method is used to free a block of memory that was allocated with Allocate (see page 481). The provided pointer will be walked up until the original pointer is found.

See Also

Allocate (see page 481)

2.10.1.6.1.2.2 - CMemoryAllocator::Deallocate Method

Frees a block of memory.

C++

```
static void Deallocate(IN TOA void* pvoidBlock);
```

Parameters

Parameters	Description
IN TOA void* pvoidBlock	A pointer to the memory block to free.

Description

This method is used to free a block of memory that was allocated with Allocate (see page 481). The provided pointer will be walked up until the original pointer is found.

See Also

Allocate (see page 481)

2.10.1.6.1.3 - CMemoryAllocator::DisableMemoryStatistics Method

Disables memory statistics acquisition.

C++

```
static void DisableMemoryStatistics();
```

Description

When memory statistics are enabled at compile time this method disables the acquisition of memory statistics at runtime.

By disabling statistics, it is possible to control the scope or the parts of the software to diagnose.

By default runtime acquisition of memory statistics is enabled.

See Also

EnableMemoryStatistics (see page 482)

2.10.1.6.1.4 - CMemoryAllocator::EnableMemoryStatistics Method

Enables memory statistics acquisition.

C++

```
static void EnableMemoryStatistics();
```

Description

When memory statistics are enabled at compile time this method enables the acquisition of memory statistics at runtime. By enabling statistics, it is possible to control the scope or the parts of the software to diagnose. By default runtime acquisition of memory statistics is enabled.

See Also

[DisableMemoryStatistics](#) (see page 482)

2.10.1.6.1.5 - CMemoryAllocator::EnumMemoryBlocks Method

Enumerates through the currently allocated list of memory blocks.

C++

```
static void EnumMemoryBlocks(IN IMemoryBlockAccumulator* pAccumulator);
```

Parameters

Parameters	Description
IN IMemoryBlockAccumulator* pAccumulator	The accumulator which will receive the enumerated memory blocks.

Description

This method takes an IMemoryBlockAccumulator (see page 488) as only parameter. It then enumerates each currently allocated memory block by passing it to the pAccumulator's IMemoryBlockAccumulator::Accumulate (see page 488) method.

The implementor of the passed in IMemoryBlockAccumulator (see page 488) needs to adhere to certain rules to avoid threading and other issues:

1. It MUST NOT directly or indirectly allocate memory using the CMemoryAllocator (see page 480):
 - MX_NEW (see page 510)
 - MX_NEW_ARRAY (see page 510)
 - CMemoryAllocator::Allocate (see page 481)
2. It MUST NOT directly or indirectly deallocate memory using the CMemoryAllocator (see page 480):
 - MX_DELETE (see page 509)
 - MX_DELETE_ARRAY (see page 510)
 - CMemoryAllocator::Deallocate (see page 482)
3. It MUST NOT be blocking.

2.10.1.6.1.6 - CMemoryAllocator::GetMemoryBlock Method

Get a pointer to the CMemoryBlock (see page 485) associated with an allocated memory block.

C++

```
static CMemoryBlock* GetMemoryBlock(IN void* pvoidBlock);
```

Parameters

Parameters	Description
IN void* pvoidBlock	Pointer to the user-space memory block from which to retrieve the CMemoryBlock (see page 485).

Returns

CMemoryAllocator::CMemoryBlock (see page 485)* A pointer to the CMemoryBlock (see page 485) associated with the user-space memory block passed in.

Description

This method is used to retrieve a pointer to a CMemoryBlock (see page 485) associated with the passed-in user-space memory block.

2.10.1.6.1.7 - CMemoryAllocator::GetMemoryStatistics Method

Gets a structure containing current memory statistics.

C++

```
static void GetMemoryStatistics(OUT SMemoryStatistics* pstMemoryStatistics);
```

Parameters

Parameters	Description
OUT SMemoryStatistics* pstMemoryStatistics	Pointer to an existing structure of SMemoryStatistics (see page 514) type.

Description

This method puts a copy of the current memory statistics in the structure passed as parameter.

See Also

Allocate (see page 481), Deallocate (see page 482), SetMemoryStatistics (see page 484), ResetMemoryStatistics (see page 484), SMemoryStatistics (see page 514)

2.10.1.6.1.8 - CMemoryAllocator::ResetMemoryStatistics Method

Resets the structure containing current memory statistics.

C++

```
static void ResetMemoryStatistics();
```

Description

This method resets the current memory statistics to their initial values. The subsequent call to GetMemoryStatistics (see page 484) will return the statistics gathered since a last call to ResetMemoryStatistics.

If the overall statistics are important they must be saved with a prior call to GetMemoryStatistics (see page 484) and restored with a call to SetMemoryStatistics (see page 484). A typical usage of the reset feature is shown in the following code snippet.

```
// Do whatever code to get the program in the state to diagnose.
// ...

// Keep a copy of the current statistics.
CMemoryAllocator::SMemoryStatistics stOverallMemoryStats;
CMemoryAllocator::GetMemoryStatistics(&stOverallMemoryStats);

// Reset the statistics.
CMemoryAllocator::ResetMemoryStatistics();

// Perform operation to diagnose.
// ...

// Get the statistics for the operation to diagnose.
CMemoryAllocator::SMemoryStatistics stMemoryStats;
CMemoryAllocator::GetMemoryStatistics(&stMemoryStats);

// Do whatever analysis on the stats, print it out, etc...
// ...

// Restore the statistics to their original values.
CMemoryAllocator::SetMemoryStatistics(&stOverallMemoryStats);

// Continue the program.
// ...
```

See Also

GetMemoryStatistics (see page 484), SetMemoryStatistics (see page 484), SMemoryStatistics (see page 514)

2.10.1.6.1.9 - CMemoryAllocator::SetMemoryStatistics Method

Sets the structure containing current memory statistics.

C++

```
static void SetMemoryStatistics(IN SMemoryStatistics* pstMemoryStatistics);
```

Parameters

Parameters	Description
IN SMemoryStatistics* pstMemoryStatistics	Pointer to an existing structure of SMemoryStatistics (see page 514) type.

Description

This method sets the current memory statistics to the statistics in the structure passed as parameter.

See Also

[GetMemoryStatistics](#) (see page 484), [ResetMemoryStatistics](#) (see page 484), [SMemoryStatistics](#) (see page 514)

2.10.1.6.2 - Nested Types

2.10.1.6.2.1 - CMemoryAllocator::PFNAllocateFunction Nested Type

```
typedef GO void* (* PFNAllocateFunction)(IN size_t uSize);
```

Returns

void* A pointer to the requested memory block.

Description

This is a type describing a function pointer used to allocate a block of memory according to a specified size.

2.10.1.6.2.2 - CMemoryAllocator::PFNDeallocateFunction Nested Type

```
typedef void (* PFNDeallocateFunction)(IN TOA void* pvoidBlock);
```

Description

This type describes a function pointer to a function used to deallocate a block of memory.

2.10.1.7 - CMemoryAllocator::CMemoryBlock Class

Class used to store an allocated memory block.

Class Hierarchy

C++

```
class CMemoryBlock;
```

Description

This class holds an allocated memory block and its header. The `m_blockHeader` variable holds the header of memory block and the `m_uStartOfBlock` variable marks the beginning of the user-accessible block.

The public interface that is provided is safe to use by outsiders of this class provided they don't try to modify the pointer returned by `GetPointer` (see page 486).

See Also

[SMemoryBlockExtraInfo](#) (see page 512)

Methods

Method	Description
◆ GetFilename (see page 486)	Returns the filename in which the memory block was allocated.
◆ GetLineNumber (see page 486)	Returns the line number at which the memory block was allocated.
◆ GetPointer (see page 486)	Returns the pointer of the user-block.
◆ GetSize (see page 486)	Returns the size of the allocated memory block.
◆ GetType (see page 487)	Returns a zero-terminated string containing the type of the allocated memory block.
◆ IsPooled (see page 487)	Returns whether the memory block is currently marked as being pooled in a memory pool.
◆ IsTemporarilyUntrackedFlagSet (see page 487)	Returns whether the memory block was temporarily marked as untracked.
◆ SetPooled (see page 487)	Sets whether the memory block is marked as a pooled memory block.
◆ SetTemporarilyUntrackedFlag (see page 487)	Sets whether the memory block is temporarily marked as untracked.

Legend



2.10.1.7.1 - Methods

2.10.1.7.1.1 - CMemoryAllocator::CMemoryBlock::GetFilename Method

Returns the filename in which the memory block was allocated.

C++

```
char const* GetFilename() const;
```

Returns

char const* Null-terminated string containing the filename in which the CMemoryBlock (see page 485) was allocated.

Description

This method returns a null-terminated string representing the filename of the file in which the CMemoryBlock (see page 485) was allocated.

2.10.1.7.1.2 - CMemoryAllocator::CMemoryBlock::GetLineNumber Method

Returns the line number at which the memory block was allocated.

C++

```
uint16_t GetLineNumber() const;
```

Returns

uint16_t (see page 85) The line number at which this memory block was allocated.

Description

This method returns the line number at which this memory block was allocated.

2.10.1.7.1.3 - CMemoryAllocator::CMemoryBlock::GetPointer Method

Returns the pointer of the user-block.

C++

```
void* GetPointer();
```

Returns

void* The user-space pointer to the memory location returned by CMemoryAllocator::Allocate (see page 481).

Description

Method returning the pointer to the user-space memory location of the CMemoryBlock (see page 485).

2.10.1.7.1.4 - CMemoryAllocator::CMemoryBlock::GetSize Method

Returns the size of the allocated memory block.

C++

```
size_t GetSize() const;
```

Returns

size_t The number of bytes that this memory block reserves for user accessible memory.

Description

This method returns the size in bytes that this memory block reserves for user accessible memory.

2.10.1.7.1.5 - CMemoryAllocator::CMemoryBlock::GetType Method

Returns a zero-terminated string containing the type of the allocated memory block.

C++

```
char const* GetType() const;
```

Returns

char const* Null-terminated string containing the type of the object contained in the allocated memory block.

Description

This method returns a null-terminated string containing the type of the object contained in the allocated memory block.

2.10.1.7.1.6 - CMemoryAllocator::CMemoryBlock::IsPooled Method

Returns whether the memory block is currently marked as being pooled in a memory pool.

C++

```
bool IsPooled() const;
```

Returns

bool True if the "Is Pooled" flag is set for this memory block. False otherwise.

Description

This method returns whether the "Is Pooled" flag is set for this memory block.

2.10.1.7.1.7 - CMemoryAllocator::CMemoryBlock::IsTemporarilyUntrackedFlagSet Method

Returns whether the memory block was temporarily marked as untracked.

C++

```
bool IsTemporarilyUntrackedFlagSet() const;
```

Returns

bool True if the "Temporarily Untracked" flag is set for this memory block. False otherwise.

Description

This method returns whether the "Temporarily Untracked" flag is set for this memory block.

2.10.1.7.1.8 - CMemoryAllocator::CMemoryBlock::SetPooled Method

Sets whether the memory block is marked as a pooled memory block.

C++

```
void SetPooled(IN bool bIsPooled);
```

Parameters

Parameters	Description
IN bool bIsPooled	The new value for the "Is Pooled" flag.

Description

This method is used to mark the memory block when it is put in a memory pool. This allows the user to filter pooled memory blocks as they are enumerated from the CMemoryAllocator::EnumMemoryBlock method.

2.10.1.7.1.9 - CMemoryAllocator::CMemoryBlock::SetTemporarilyUntrackedFlag Method

Sets whether the memory block is temporarily marked as untracked.

C++

```
void SetTemporarilyUntrackedFlag(IN bool bIsSet);
```

Parameters

Parameters	Description
IN bool bIsSet	The new value for the "Temporarily Untracked" flag.

Description

This method is used to mark the memory block as Temporarily Untracked. This allows the user to filter existing memory blocks as they are enumerated from the CMemoryAllocator::EnumMemoryBlock method.

2.10.1.8 - CMemoryAllocator::IMemoryBlockAccumulator Class

Interface defining an accumulator of CMemoryBlocks.

Class Hierarchy

C++

```
class IMemoryBlockAccumulator;
```

Description

This interface defines a simple CMemoryBlock (see page 485) accumulator. The implementation of this interface MUST be prepared to receive multiple successive calls to the Accumulate (see page 488) method, accept and process each memory block in an implementation dependent way.

See Also

CMemoryBlock (see page 485), CMemoryBlockList::Enumerate

Methods

Method	Description
◆ A Accumulate (see page 488)	Notification function which receives the memory blocks being enumerated.

Legend

◆	Method
A	abstract

2.10.1.8.1 - Methods

2.10.1.8.1.1 - CMemoryAllocator::IMemoryBlockAccumulator::Accumulate Method

Notification function which receives the memory blocks being enumerated.

C++

```
virtual void Accumulate(INOUT CMemoryBlock* pMemoryBlock) = 0;
```

Parameters

Parameters	Description
INOUT CMemoryBlock* pMemoryBlock	A pointer to the memory blocks being accumulated.

Description

This notification function is the callback function called by the CMemoryAllocator::EnumMemoryBlocks (see page 483) method while enumerating through the list of currently allocated memory blocks.

See Also

CMemoryBlock (see page 485), CMemoryAllocator::EnumMemoryBlocks (see page 483)

2.10.1.9 - CMutex Class

Basic synchronization object.

Class Hierarchy

```
CMutex
```

C++

```
class CMutex;
```

Description

The mutex is a basic synchronization object. A thread can lock the mutex multiple times once it has already entered the protected section. The thread must unlock the mutex the same number of times for it to be released. Mutexes are mostly used to protect part of code that require mutual exclusion (also referred to as critical section).

Location

Kernel/CMutex.h

See Also

CBinarySemaphore (see page 467), CCountingSemaphore (see page 470)

Constructors

Constructor	Description
CMutex (see page 489)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CMutex (see page 489)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 Lock (see page 490)	Locks the mutex.
 TryLock (see page 490)	Tries to lock the mutex.
 Unlock (see page 490)	Unlocks the mutex.

Legend

	Method
---	--------

2.10.1.9.1 - Constructors

2.10.1.9.1.1 - CMutex::CMutex Constructor

Constructor.

C++

```
CMutex();
```

Description

Constructor.

2.10.1.9.2 - Destructors

2.10.1.9.2.1 - CMutex::~CMutex Destructor

Destructor.

C++

```
virtual ~CMutex();
```

Description

Destructor.

2.10.1.9.3 - Methods

2.10.1.9.3.1 - CMutex::Lock Method

Locks the mutex.

C++

```
void Lock() const;
```

Description

Locks the mutex. If the mutex is already locked by another thread, the call will block until the mutex is released. The mutex can be locked multiple times by the same thread but must be unlocked the same number of times.

2.10.1.9.3.2 - CMutex::TryLock Method

Tries to lock the mutex.

C++

```
bool TryLock() const;
```

Returns

True if the mutex was locked.

Description

Tries to lock the mutex. TryLock has the same behavior as Lock (see page 490), except that it returns immediately with false instead of blocking if the mutex is locked by another thread.

2.10.1.9.3.3 - CMutex::Unlock Method

Unlocks the mutex.

C++

```
void Unlock() const;
```

Description

Unlocks the mutex. A thread can only unlock a mutex it owns.

2.10.1.10 - COsVersion Class

This class gives access to operating system version.

Class Hierarchy

```
COsVersion
```

C++

```
class COsVersion;
```

Description

This class gets the operating system version at run time.

Location

Kernel/COsVersion.h

Constructors

Constructor	Description
 COsVersion (see page 491)	Constructor.

Legend



Destructors

Destructor	Description
 ~COsVersion (see page 491)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 GetOsVersion (see page 491)	Retrieves the version of the operating system.

Legend

	Method
---	--------

2.10.1.10.1 - Constructors**2.10.1.10.1.1 - COsVersion::COsVersion Constructor**

Constructor.

C++

```
COsVersion();
```

Description

Default constructor.

2.10.1.10.2 - Destructors**2.10.1.10.2.1 - COsVersion::~COsVersion Destructor**

Destructor.

C++

```
virtual ~COsVersion();
```

Description

Destructor.

2.10.1.10.3 - Methods**2.10.1.10.3.1 - COsVersion::GetOsVersion Method**

Retrieves the version of the operating system.

C++

```
static EOsversion GetOsVersion();
```

Returns

The OS version number if it can be determined.

Description

This method uses OS specific methods to get the version at run time.

2.10.1.11 - CThread Class

Allows the creation of a new thread of execution.

Class Hierarchy

 CThread

C++

```
class CThread;
```

Description

This class allows the creation of a new thread of execution.

It can control the start-up and synchronize on the termination while fetching an optional return code. The execution priority is adjustable on 5 distinctive levels (which are all static and predictable).

This class also supports registration of thread specific data (TSD), as long as a unique thread Id is created even for non-CThread threads. It also offers global methods to retrieve the CThread object of the currently executing thread (if, of course, it is a derivative of CThread).

The interface of CThread is identical and the behavior of the threads are predictable for every supported operating system.

Warning

CThread implementation on the Nucleus platform is based on the NUCLEUS POSIX layer. This layer is not compatible with the standard NUCLEUS task management functions. M5T code behavior is not defined when it is called from a non POSIX thread context.

Location

Kernel/CThread.h

Example

Here is an example of how to create a high priority thread of execution.

```
// Starts a new high priority thread with the default stack size.
CThread t;
mxt_opaque opqResult;
t.StartThread(ResolverThreadEntry,
              MX_VOIDPTR_TO_OPQ(this),
              "ResolverThread",
              0,
              CThread::eHIGH);

// Wait for thread termination.
t.Join(&opqResult);
```

Constructors

Constructor	Description
◆ CThread (see page 493)	Constructor.

Legend

◆	Method
---	--------

Destructors

Destructor	Description
◆ ~CThread (see page 493)	Destructor.

Legend

◆	Method
▼	virtual

Operators

Operator	Description
◆ != (see page 500)	Different than operator.
◆ == (see page 500)	Comparison operator.

Legend

◆	Method
---	--------

Methods

Method	Description
◆ CreateKey (see page 493)	Creates a new TSD key.
◆ DeleteKey (see page 494)	Deletes the specified TSD key.
◆ FinalizeTsd (see page 494)	Finalizes the TSD.
◆ GetCurrentId (see page 494)	Gets the application ID of the current thread of execution.

• GetId (see page 495)	Gets the application ID of this thread.
• GetName (see page 495)	Gets the name of the thread.
• GetNativeThreadId (see page 495)	Gets the native thread ID.
• GetSpecific (see page 495)	Gets the data for the specified key.
• GetState (see page 496)	Gets the current state of the thread.
• GetThreadSelf (see page 496)	Gets the address of the current thread.
• GetThreadStackInfo (see page 497)	Gets the stack information of a specific CThread.
• InitializeTsd (see page 498)	Initializes the TSD.
• IsCurrentThread (see page 498)	Checks if thread is current thread of execution.
• Join (see page 498)	Waits for thread termination.
• ResetThreadStackInfo (see page 499)	Resets the stack information for the current CThread.
• SetSpecific (see page 499)	Sets new data for specified key.
• StartThread (see page 499)	Starts a new thread.

Legend**2.10.1.11.1 - Constructors****2.10.1.11.1.1 - CThread::CThread Constructor**

Constructor.

C++

```
CThread();
```

Description

The CThread object constructor does merely nothing except initializing the internal members, and the OS-specific support for TSD (which is done the first time only).

After its creation, a CThread object has no unique ID, no name, etc. Its state is eNOT_STARTED. To start a real thread, the method StartThread (see page 499) should be used.

See Also

[CThread::StartThread](#) (see page 499)

2.10.1.11.2 - Destructors**2.10.1.11.2.1 - CThread::~CThread Destructor**

Destructor.

C++

```
virtual ~CThread();
```

Description

The destructor of the CThread (see page 491) object will wait for the termination of the thread, and then set the internal state to eDELETED.

2.10.1.11.3 - Methods**2.10.1.11.3.1 - CThread::CreateKey Method**

Creates a new TSD key.

C++

```
static mxt_result CreateKey(IN PFNKeyDeletion pfnKeyDeletion, OUT mxt_tsdkey & rTsdKey);
```

Parameters

Parameters	Description
IN PFNKeyDeletion pfnKeyDeletion	The address of a function to be called upon thread termination. This function will receive in parameter the specific datum registered by that thread that just terminated. If no datum was registered (or set to a datum of 0), this function won't be called. This parameter can safely be NULL, in which case no deletion function will be called upon thread termination.
OUT mxt_tsdkkey & rTsdKey	A reference to a variable (usually of global scope) that will receive the new key. In case of error, that parameter remains unchanged.

Returns

The possible error code is resFE_MITOSFW_KERNEL_OUT_OF_MEMORY, if there was a lack of memory to create a TSD area.

Description

This static method creates a new thread specific data (TSD) key with global scope. This key can then be used to register thread specific data using method SetSpecific (see page 499).

See Also

CThread::SetSpecific (see page 499), CThread::GetSpecific (see page 495), CThread::DeleteKey (see page 494)

2.10.1.11.3.2 - CThread::DeleteKey Method

Deletes the specified TSD key.

C++

```
static mxt_result DeleteKey(IN mxt_tsdkkey tsdkkey);
```

Parameters

Parameters	Description
IN mxt_tsdkkey tsdkkey	The key to be deleted, as returned by method CreateKey (see page 493).

Returns

The possible error code is: resFE_MITOSFW_KERNEL_INVALID_TSD_KEY The specified TSD key does not exist or has already been deleted.

Description

This static method deletes a TSD key created by method CreateKey (see page 493).

If there are any data still referenced by that key, they are lost, that is, they become unreachable by the thread that have added it (method GetSpecific (see page 495) returns an error). Also, the deletion function is not called either nor later.

See Also

CThread::CreateKey (see page 493)

2.10.1.11.3.3 - CThread::FinalizeTsd Method

Finalizes the TSD.

C++

```
static void FinalizeTsd();
```

2.10.1.11.3.4 - CThread::GetCurrentId Method

Gets the application ID of the current thread of execution.

C++

```
static unsigned GetCurrentId();
```

Returns

The unique application ID of the current thread of execution, or 0 if there was a lack of memory to create a TSD area.

Description

This static method returns a non-zero unique application ID associated to the current thread of execution. The ID will not change for all

of the thread's life. This method can be used on any type of thread, even non-CThread (see page 491). It is possible that two threads started by two different processes have the same thread ID since this value is not OS related.

Note that if more than 2³² threads are created, the unique ID generator will wrap and the behavior is unpredictable.

2.10.1.11.3.5 - CThread::GetId Method

Gets the application ID of this thread.

C++

```
unsigned GetId() const;
```

Returns

The unique application ID of the thread of the CThread (see page 491) object. A value of 0 means the thread was never started (StartThread (see page 499) never called), and if the thread has terminated, it returns the application ID of the last thread.

Description

Gets the unique application ID of the thread of this CThread (see page 491) object.

See Also

CThread::GetCurrentId (see page 494)

2.10.1.11.3.6 - CThread::GetName Method

Gets the name of the thread.

C++

```
const char* GetName() const;
```

Returns

A pointer to a null-terminated character string which contains the name of the current (or last created) thread. An empty string is returned if no thread was started yet for this CThread (see page 491) object.

Description

Gets the name of the thread, which was given to StartThread (see page 499).

2.10.1.11.3.7 - CThread::GetNativeThreadId Method

Gets the native thread ID.

C++

```
mxt_result GetNativeThreadId(OUT int & nNativeThreadId);
```

Parameters

Parameters	Description
rnNativeThreadId	Reference to the integer containing the native thread ID value.

Returns

resS_OK: The method succeeded. resFE_FAILED: The thread ID value is invalid. resFE_NOT_IMPLEMENTED: The method is not implemented on this operating system.

Description

Gets the native thread ID value. The native thread ID is given by the operating system to be able to differentiate each unique thread. Under Linux, this value matches its thread ID returned by the 'ps' command.

2.10.1.11.3.8 - CThread::GetSpecific Method

Gets the data for the specified key.

C++

```
static mxt_result GetSpecific(IN mxt_tsdkkey tsdKey, OUT mxt_opaque& ropqDatum);
```

Parameters

Parameters	Description
IN mxt_tsdkkey tsdkKey	The TSD key returned by method CreateKey (see page 493).
OUT mxt_opaque& ropqDatum	A reference to a variable that will receive the datum set by method SetSpecific (see page 499), for the current thread of execution. If no datum was set for this key, the datum returned is 0, with no error.

Returns

Possible error codes are: resFE_MITOSFW_KERNEL_INVALID_TSD_KEY The specified TSD key does not exist or has been deleted. resFE_MITOSFW_KERNEL_OUT_OF_MEMORY There was a lack of memory to create a TSD area.

Description

This static method reads the datum specific to the current thread of execution, as set by method SetSpecific (see page 499). If no datum was set with SetSpecific (see page 499), the returned datum has a value of 0.

Note that the datum of a thread other than the current thread can not be read nor modified. A specific datum can be set on any thread, even non-CThread (see page 491).

See Also

CThread::SetSpecific (see page 499), CThread::CreateKey (see page 493)

2.10.1.11.3.9 - CThread::GetState Method

Gets the current state of the thread.

C++

```
EState GetState() const;
```

Returns

The current state of the CThread (see page 491) object.

Description

Gets the current state of the CThread (see page 491) object.

The possible states are:

- eNOT_STARTED After the creation of the CThread (see page 491) object (before calling StartThread (see page 499)), after a call to StartThread (see page 499) that failed, and after a successful call to Join (see page 498).
- eSTARTING After a successful call to StartThread (see page 499), but before the thread begins its execution.
- eACTIVE The thread is currently executing its thread entry function.
- eTERMINATED The thread entry function has exited (the thread execution is completed), but a successful call to Join (see page 498) as not been done yet.
- eDELETED The CThread (see page 491) object destructor was called. NOTE: When working from the standard heap memory (malloc/free), this state should never be seen, and it would most certainly indicate a bug.

2.10.1.11.3.10 - CThread::GetThreadSelf Method

Gets the address of the current thread.

C++

```
static CThread* GetThreadSelf();
```

Returns

The address of the CThread (see page 491) object that owns the current thread of execution, or NULL if the current thread is not a CThread (see page 491).

Description

This static method returns the address of the current CThread (see page 491) object.

2.10.1.11.3.11 - GetThreadStackInfo

2.10.1.11.3.11.1 - CThread::GetThreadStackInfo Method

Gets the stack information of a specific CThread (see page 491).

C++

```
static mxt_result GetThreadStackInfo(IN const char* pszThreadName, OUT SThreadStackInfo* pstStackInfo);
```

Parameters

Parameters	Description
IN const char* pszThreadName	A null-terminated character string representing the name of a CThread (see page 491).
OUT SThreadStackInfo* pstStackInfo	Pointer to a SThreadStackInfo (see page 514) structure.

Returns

resS_OK: The method succeeded. resFE_FAIL: A CThread (see page 491) of the specified name was not found.
resFE_INVALID_ARGUMENT: The pstStackInfo parameter is NULL.

Description

This method gets the stack information for the CThread (see page 491) that has the name specified in pszThreadName. If found, it fills the stack usage structure pointed by pstStackInfo with the current stack usage information. If no CThread (see page 491) name in the system correspond to the name stored in the method return and the content pointed by pstStackInfo is left as is.

In the case where more than one thread have the same name, only the stack information of the first occurrence found is returned. The stack information of other CThread (see page 491) can always be retrieved with the overloaded version of GetThreadStackInfo.

See Also

SThreadStackInfo (see page 514), CThread::ResetThreadStackInfo (see page 499),
MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)

2.10.1.11.3.11.2 - CThread::GetThreadStackInfo Method

Gets the stack information for all CThread (see page 491).

C++

```
static mxt_result GetThreadStackInfo(IN unsigned int uStackInfoCapacity, OUT SThreadStackInfo* pstStackInfo, OUT
unsigned int* puStackInfoSize);
```

Parameters

Parameters	Description
IN unsigned int uStackInfoCapacity	Capacity of the pstStackInfo array.
OUT SThreadStackInfo* pstStackInfo	Pointer to the first element in an array of SThreadStackInfo (see page 514) that will contain the stack information on return.
OUT unsigned int* puStackInfoSize	Pointer to a unsigned int that will contain the number of elements stored in the array. Or, the number of CThread (see page 491) that currently exist.

Returns

resS_OK: The method succeeded. resFE_INVALID_ARGUMENT: The puStackInfoSize parameter is NULL, or uStackInfoCapacity is greater than 0 while pstStackInfo is NULL.

Description

This method gets the stack information for all CThread (see page 491) that currently exist. If the pstStackInfo parameter is NULL, this method only stores the current number of CThread (see page 491) in puStackInfoSize. Otherwise, it fills pstStackInfo with the stack usage information of up to uStackInfoCapacity CThread (see page 491) objects.

See Also

SThreadStackInfo (see page 514), CThread::ResetThreadStackInfo (see page 499),
MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)

2.10.1.11.3.11.3 - CThread::GetThreadStackInfo Method

Gets the stack information of the current CThread (see page 491).

C++

```
static mxt_result GetThreadStackInfo(OUT SThreadStackInfo* pstStackInfo);
```

Parameters

Parameters	Description
OUT SThreadStackInfo* pstStackInfo	Pointer to a SThreadStackInfo (see page 514) structure.

Returns

resS_OK: The method succeeded. resFE_FAIL: The current thread of execution is not a CThread (see page 491).
 resFE_INVALID_ARGUMENT: The pstStackInfo parameter is NULL.

Description

This method gets the stack information for the currently running CThread (see page 491). It fills the pstStackInfo structure with the peak stack usage and name of the CThread (see page 491).

See Also

SThreadStackInfo (see page 514), CThread::ResetThreadStackInfo (see page 499),
 MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)

2.10.1.11.3.12 - CThread::InitializeTsd Method

Initializes the TSD.

C++

```
static mxt_result InitializeTsd();
```

2.10.1.11.3.13 - CThread::IsCurrentThread Method

Checks if thread is current thread of execution.

C++

```
bool IsCurrentThread() const;
```

Returns

true if the CThread (see page 491) object owns the current thread of execution, or false otherwise.

Description

This method confirms that the CThread (see page 491) object matches the current thread of execution. The comparison is done on the thread's unique application ID.

2.10.1.11.3.14 - CThread::Join Method

Waits for thread termination.

C++

```
mxt_result Join(OUT mxt_opaque * popqExitCode = NULL);
```

Parameters

Parameters	Description
OUT mxt_opaque * popqExitCode = NULL	The address of an opaque variable to receive the exit code of the thread, that is, the code returned by the entry function given to StartThread (see page 499). If no thread of execution was created yet with that CThread (see page 491) object, the return code is set to 0. This parameter can safely be NULL, in which case the exit code will be ignored.

Returns

The possible failure result is: resFE_MITOSFW_KERNEL_DEADLOCK_JOIN This error code is returned when a thread of execution calls Join on its own CThread (see page 491) object, instead of dead locking.

Description

Wait for the termination of the thread of execution associated to this CThread (see page 491) object, and set the state to eNOT_STARTED. If the current status is already eNOT_STARTED, the method returns immediately with no error. The exit code can be collected if the address of an opaque variable is given in parameter. More than one thread of execution can call Join on the same CThread (see page 491) object. They will correctly wait for the termination of that thread, but the order of their waking up is unpredictable. If a thread deletes its own CThread (see page 491) object, another thread calling Join will cause either a segmentation fault or a deadlock.

2.10.1.11.3.15 - CThread::ResetThreadStackInfo Method

Resets the stack information for the current CThread (see page 491).

C++

```
static mxt_result ResetThreadStackInfo();
```

Returns

resS_OK: The method succeeded. resFE_FAIL: The current thread of execution is not a CThread (see page 491).
resFE_INVALID_STATE: m_puStartOfStack is NULL for the current CThread (see page 491).

Description

This method resets the stack informations for the current CThread (see page 491). It resets the peak stack usage by filling the stack from it's current location to it's end with a known pattern.

Note that it is impossible to entirely reset the stack with this pattern as this method and it's members use stack space. The stack will be reset from the closest possible location of the stack pointer after entering this method.

See Also

CThread::GetThreadStackInfo (see page 497), MXD_THREAD_STACK_INFO_ENABLE_SUPPORT (see page 308)

2.10.1.11.3.16 - CThread::SetSpecific Method

Sets new data for specified key.

C++

```
static mxt_result SetSpecific(IN mxt_tsdkKey tsdKey, IN mxt_opaque opqDatum);
```

Parameters

Parameters	Description
IN mxt_tsdkKey tsdKey	The TSD key returned by method CreateKey (see page 493).
IN mxt_opaque opqDatum	Any opaque value can be set here. That value will be passed unchanged to the registered key deletion function upon thread termination. A value of 0 removes the specific datum from the current thread.

Returns

The possible error codes are: resFE_MITOSFW_KERNEL_INVALID_TSD_KEY The specified TSD key does not exist or has been deleted. resFE_MITOSFW_KERNEL_OUT_OF_MEMORY There was a lack of memory to create a TSD area.

Description

This static method sets or overwrites the specific datum associated to the TSD key, for the current thread. This datum can later be read using method GetSpecific (see page 495). The specific datum can also be cleared by calling SetSpecific with a value of 0, which will also free the associated memory.

Note that the datum of a thread other than the current thread can not be read nor modified. A specific datum can be set on any thread, even non-CThread (see page 491).

See Also

CThread::GetSpecific (see page 495), CThread::CreateKey (see page 493)

2.10.1.11.3.17 - CThread::StartThread Method

Starts a new thread.

C++

```
mxt_result StartThread(IN PFNEntryPoint pfnThreadEntry, IN mxt_opaque opqParam, IN const char * pszName = NULL, IN uint32_t uStackSize = 0, IN EPriority ePriority = eNORMAL);
```

Parameters

Parameters	Description
IN PFNEntryPoint pfnThreadEntry	The address of the function to be called by the new thread of execution. This function will receive as its first argument the value of opqParam. This parameter can safely be NULL, which will effectively create a thread, but with a very short life.

IN mxt_opaque opqParam	An opaque parameter to pass unchanged to the function that parameter pfnThreadEntry refers to.
IN const char * pszName = NULL	The address of a null-terminated character string which contains the name of the thread. This parameter can safely be NULL, and a default name will be assigned. The thread name is currently useful only for debugging. When unspecified, a default value of NULL is passed. If its length is greater than g_uTHREAD_NAME_MAX_SIZE (see page 522), then it will be truncated and a NULL character will be placed at the end.
IN uint32_t uStackSize = 0	This parameter specifies the preferred stack size in bytes, and is specific to the underlying operating system. Linux - This parameter is ignored, and the stack grows automatically. Windows - This is the initial stack size, then it grows automatically. VxWorks - The real stack size. Symbian - The real stack size. When unspecified, a default value of 0 is passed. Except for Linux and Solaris, this means that the default value of the Framework is used, which is defined by the configuration macro MXD_DEFAULT_THREAD_STACK_SIZE (see page 291).
IN EPriority ePriority = eNORMAL	The priority to be assigned to the new thread of execution. There are currently 5 levels available through the CThread (see page 491) interface: eLOWEST, eLOW, eNORMAL, eHIGH, eHIGHEST. When unspecified, a default value of eNORMAL is passed.

Returns

The result of the thread creation. The possible error codes are: resFE_MITOSFW_KERNEL_SET_THREAD_PRIORITY The new thread of execution was successfully created but its priority could not be adjusted to the specified one.
 resFE_MITOSFW_KERNEL_THREAD_CREATION The new thread of execution could not be created.

Description

This method starts a new thread of execution, using the specified parameters.

It can be called many times on the same CThread (see page 491) object, but can start only one thread at a time. It waits for the termination of the previous thread of execution.

When this function returns, the state of the CThread (see page 491) object will be eSTARTING or eACTIVE, depending on the priority of the new thread or what remains in the time slice (for equal priority thread). The state is set to eACTIVE soon after the beginning of the new thread. If the new thread could not be started (resFE_MITOSFW_KERNEL_THREAD_CREATION), the state will remain eNOT_STARTED.

2.10.1.11.4 - Operators

2.10.1.11.4.1 - CThread::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CThread & rThread) const;
```

Parameters

Parameters	Description
IN const CThread & rThread	A reference to the other CThread (see page 491) object to compare with.

Returns

True if both CThread (see page 491) objects have a different unique ID, or false otherwise.

Description

This method reverts the answer returned by the operator ==.

See Also

CThread::operator==

2.10.1.11.4.2 - CThread::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CThread & rThread) const;
```

Parameters

Parameters	Description
IN const CThread & rThread	A reference to the other CThread (see page 491) object to compare with.

Returns

true if both CThread (see page 491) object have the same unique ID, or false otherwise.

Description

This method compares two CThread (see page 491) objects to see if they have the same unique ID and are, in fact, the same thread of execution.

2.10.1.11.5 - Nested Types

2.10.1.11.5.1 - CThread::mxt_tsdkkey Nested Type

A TSD key.

C++

```
typedef unsigned mxt_tsdkkey;
```

2.10.1.11.5.2 - CThread::PFNEntryPoint Nested Type

Function pointer that is used for the StartThread (see page 499) method.

C++

```
typedef mxt_opaque (* PFNEntryPoint)(IN mxt_opaque opq);
```

2.10.1.11.5.3 - CThread::PFNKeyDeletion Nested Type

Function pointer that is used for when thread terminates.

C++

```
typedef void (* PFNKeyDeletion)(IN mxt_opaque opq);
```

2.10.1.12 - CTimer Class

Class implementing timers.

Class Hierarchy

CTimer

C++

```
class CTimer;
```

Description

The CTimer class is the class implementing timers. There are two types of timers available: cyclic timers and one shot timers.

One shot timers wait for the specified time in ms and return. Cyclic timers are timers that wait for a specified time, and when the wait is over the time wait starts over. Every time the application calls a wait on a cyclic timer, it verifies if the time passed between calls to wait. If it is less than the timer, the wait waits for the remaining time. If it is greater, the timer can either set the current time in the wait so that the next call will work properly or it will exit without waiting until the overrun is corrected.

Constructors

Constructor	Description
~CTimer (see page 502)	Constructor.

Legend

	Method
---	--------

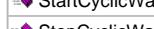
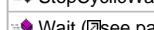
Destructors

Destructor	Description
~CTimer (see page 502)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 CyclicWait (see page 502)	Waits for the reamining time of the cycle.
 GetSystemUpTimeMs (see page 503)	Gets the number of ms since system start-up.
 StartCyclicWait (see page 503)	Starts the cyclic wait.
 StopCyclicWait (see page 503)	Stops the cyclic timer.
 Wait (see page 503)	Waits for the specified number of ms.

Legend

	Method
--	--------

2.10.1.12.1 - Constructors**2.10.1.12.1.1 - CTimer::CTimer Constructor**

Constructor.

C++

```
CTimer();
```

Description

Constructor.

2.10.1.12.2 - Destructors**2.10.1.12.2.1 - CTimer::~CTimer Destructor**

Destructor.

C++

```
virtual ~CTimer();
```

Description

Destructor.

2.10.1.12.3 - Methods**2.10.1.12.3.1 - CTimer::CyclicWait Method**

Waits for the reamining time of the cycle.

C++

```
bool CyclicWait(IN bool bAllowCycleLost = true);
```

Parameters

Parameters	Description
IN bool bAllowCycleLost = true	Will be true if cycles can be lost, false otherwise.

Returns

True if the cyclic timer is active, false otherwise.

Description

Waits for the remaining time left of the period. If the time between calls to CyclicWait is greater than the period then the behavior is dependant on the bAllowCycleLost flag. If the flag is true, the timer is reset at the missing time frame value. If set to false, it will exit without waiting until the overrun is corrected.

2.10.1.12.3.2 - CTimer::GetSystemUpTimeMs Method

Gets the number of ms since system start-up.

C++

```
static uint64_t GetSystemUpTimeMs();
```

Returns

The time in ms since system start-up.

Description

Gets the time elapsed since system start-up in ms.

Warning

GetSysUpTime MUST be called at least once every 49 days to work properly.

2.10.1.12.3.3 - CTimer::StartCyclicWait Method

Starts the cyclic wait.

C++

```
bool StartCyclicWait(IN uint64_t uMs);
```

Parameters

Parameters	Description
IN uint64_t uMs	Time period of one cycle in ms.

Returns

True if the cyclic timer was successfully started, false otherwise.

Description

Starts a cyclic timer and set the start time.

2.10.1.12.3.4 - CTimer::StopCyclicWait Method

Stops the cyclic timer.

C++

```
void StopCyclicWait();
```

Description

Stops a cyclic timer.

2.10.1.12.3.5 - CTimer::Wait Method

Waits for the specified number of ms.

C++

```
static void Wait(IN uint64_t uMs);
```

Parameters

Parameters	Description
IN uint64_t uMs	Time to wait in ms.

Description

Waits for the specified time in milliseconds.

Warning

The real time elapsed during the call to wait will be as precise as the platform can be. As such a wait time of 1 millisecond may wait longer, depending on the scheduler of the operating system.

2.10.2 - Enumerations

This section documents the enumerations of the Sources/Kernel folder.

Enumerations

Enumeration	Description
EInitValue (See page 504)	Defines the possible initial states of the binary semaphore.
EInitValue (See page 504)	Defines the possible initial states of the binary semaphore.
EFlags (See page 504)	Defines the possible flags to use when opening a file.
EMode (See page 505)	Defines the possible file modes.
ESeekPositions (See page 505)	Defines the possible available seek positions.
EMemoryBlockFlags (See page 505)	This enumeration describes the different bits in the m_uFlags (See page 513) bitfield.
EOsVersion (See page 505)	Defines supported OS versions.
EPriority (See page 506)	Define 5 standard levels of priority.

2.10.2.1 - CBinarySemaphore::EInitValue Enumeration

Defines the possible initial states of the binary semaphore.

C++

```
enum EInitValue {
    eACQUIRED = 0,
    eFREE = 1
};
```

Members

Members	Description
eACQUIRED = 0	Semaphore is aquired.
eFREE = 1	Semaphore is free.

2.10.2.2 - CCountingSemaphore::EInitValue Enumeration

Defines the possible initial states of the binary semaphore.

C++

```
enum EInitValue {
    eALL_ACQUIRED = 0,
    eALL_FREE = uSEM_MAX_COUNT
};
```

Members

Members	Description
eALL_ACQUIRED = 0	Counting semaphore is set to all aquired.
eALL_FREE = uSEM_MAX_COUNT	Counting semaphore is set to all free.

2.10.2.3 - CFile::EFlags Enumeration

Defines the possible flags to use when opening a file.

C++

```
enum EFlags {
    eREAD_ONLY = O_RDONLY,
    eWRITE_ONLY = O_WRONLY,
    eREAD_WRITE = O_RDWR,
    eCREATE = O_CREAT,
    eTRUNCATE = O_TRUNC,
    eAPPEND = O_APPEND,
    eBINARY = O_BINARY,
    eTEXT = O_TEXT
};
```

Description

When opening a file, you can combine these flags to obtain the behaviour you want when accessing the file.

Members

Members	Description
eREAD_ONLY = O_RDONLY	Opens a file for reading only. No writing operations are allowed.

eWRITE_ONLY = O_WRONLY	Opens a file for writing only. No reading operations are allowed.
eREAD_WRITE = O_RDWR	Opens a file for reading and writing. This is the default flag.
eCREATE = O_CREAT	Opens a file and creates it if it does not exist. If the file already exists, it will still be opened. It cannot be used with read-only flag.
eTRUNCATE = O_TRUNC	Truncates a file upon opening. The file must be opened in write mode (read-write or write-only), otherwise open fails. The file will have a size of 0 bytes after this operation.
eAPPEND = O_APPEND	Appends every write operation to the end of file. The file must be opened with write mode (read-write or write-only), otherwise open fails.
eBINARY = O_BINARY	Opens a file in binary mode. This is the default mode.
eTEXT = O_TEXT	Opens a file in text mode. This mode is only available on Windows. It will be ignored on all other operating systems.

2.10.2.4 - CFile::EMode Enumeration

Defines the possible file modes.

C++

```
enum EMode {
    eUSER_R = S_IREAD,
    eUSER_W = S_IWRITE,
    eUSER_X = S_IEXEC,
    eUSER_RWX = S_IRWXU
};
```

Members

Members	Description
eUSER_R = S_IREAD	Read (see page 475) mode.
eUSER_W = S_IWRITE	Write (see page 477) mode.
eUSER_X = S_IEXEC	Execute mode.
eUSER_RWX = S_IRWXU	Read (see page 475)/Write (see page 477)/Execute mode.

2.10.2.5 - CFile::ESeekPositions Enumeration

Defines the possible available seek positions.

C++

```
enum ESeekPositions {
    eSEEK_SET = SEEK_SET,
    eSEEK_CUR = SEEK_CUR,
    eSEEK_END = SEEK_END
};
```

Members

Members	Description
eSEEK_SET = SEEK_SET	Offset is relative to the start of the file
eSEEK_CUR = SEEK_CUR	Offset is relative to the current position in the file.
eSEEK_END = SEEK_END	Offset is relative to the end of the file.

2.10.2.6 - CMemoryAllocator::SMemoryBlockExtraInfo::EMemoryBlockFlags Enumeration

This enumeration describes the different bits in the m_uFlags (see page 513) bitfield.

C++

```
enum EMemoryBlockFlags {
    eMEMORY_BLOCK_TEMPORARILY_UNTRACKED_FLAG = 0x0001,
    eMEMORY_BLOCK_POOLED = 0x0002
};
```

Members

Members	Description
eMEMORY_BLOCK_TEMPORARILY_UNTRACKED_FLAG = 0x0001	Flag used to mark the memory block as temporarily untracked.
eMEMORY_BLOCK_POOLED = 0x0002	Flag used to mark the memory block as pooled.

2.10.2.7 - COsVersion::EOsVersion Enumeration

Defines supported OS versions.

C++

```
enum EOsVersion {
    eNO_VERSION_FOUND,
    eWINDOWS_NT,
    eWINDOWS_XP,
    eWINDOWS_2K,
    eWINDOWS_2003_PLUS,
    eWINDOWS_VISTA,
    eLINUX_24,
    eLINUX_26,
    eSOLARIS_8,
    eSOLARIS_10,
    eUNSUPPORTED_OS
};
```

Description

Defines supported OS versions.

Members

Members	Description
eNO_VERSION_FOUND	No OS version was found
eWINDOWS_NT	OS version is Windows NT.
eWINDOWS_XP	OS version is Windows XP.
eWINDOWS_2K	OS version is Windows 2000.
eWINDOWS_2003_PLUS	OS version is Windows 2003.
eWINDOWS_VISTA	OS version is Windows Vista or more recent.
eLINUX_24	OS version is Linux 2.4 kernel.
eLINUX_26	OS version is Linux 2.6 kernel.
eSOLARIS_8	OS version is Solaris 8.
eSOLARIS_10	OS version is Solaris 10.
eUNSUPPORTED_OS	OS version is unsupported.

2.10.2.8 - CThread::EPriority Enumeration

Define 5 standard levels of priority.

C++

```
enum EPriority {
    eLOWEST,
    eLOW,
    eNORMAL,
    eHIGH,
    eHIGHEST,
    eSTARTING,
    eACTIVE,
    eTERMINATED,
    eDELETED
};
```

Members

Members	Description
eLOWEST	Lowest priority.
eLOW	Low priority.
eNORMAL	Normal priority.
eHIGH	High priority.
eHIGHEST	Highest priority.
eSTARTING	Thread is strarting.
eACTIVE	Thread is active.
eTERMINATED	Thread is terminated.
eDELETED	Thread is deleted.

2.10.3 - Functions

This section documents the functions of the Sources/Kernel folder.

Functions

Function	Description
MxChmod (see page 507)	Changes the file's mode.
MxRemove (see page 507)	Removes the specified file.
MxRename (see page 507)	Renames the specified file.
MxTraceCallOutputHandler (see page 508)	Calls the output handler.
MxTraceSetThreadMessagesPerPeriod (see page 508)	Sets the internal thread number of messages outputted per period.
MxTraceSetThreadPeriodMs (see page 509)	Sets the internal thread waiting period.

2.10.3.1 - MxChmod Function Deprecated since v2.1.5

Changes the file's mode.

C++

```
bool MxChmod(IN const char* pszPathName, IN CFile::EMode eMode);
```

Parameters

Parameters	Description
IN const char* pszPathName	Name and path of the file to change permissions.
IN CFile::EMode eMode	Mode representing the permissions to set for the file.

Returns

- true: File's permissions changed.
- false: Cannot change file's permissions. It may be already opened or does not exist.

Description

Changes the permissions on the specified file.

This function may have a different behavior on different operating systems. Please refer to the SetFileAttributes method documentation for the Windows operating system and the chmod method documentation for other operating systems.

The use of this method is deprecated in favor of the new method CFileTools::ChangeMode (see page 479)().

See Also

Kernel/CFileTools.h

2.10.3.2 - MxRemove Function Deprecated since v2.1.5

Removes the specified file.

C++

```
bool MxRemove(IN const char* pszPathName);
```

Parameters

Parameters	Description
IN const char* pszPathName	Name and path of the file to remove.

Returns

- true: File removed successfully.
- false: Cannot remove file. It may be already opened or does not exist.

Description

Removes the specified file.

The use of this method is deprecated in favor of the new method CFileTools::Remove (see page 480)().

See Also

Kernel/CFileTools.h

2.10.3.3 - MxRename Function Deprecated since v2.1.5

Renames the specified file.

C++

```
bool MxRename(IN const char* pszOldPathName, IN const char* pszNewPathName);
```

Parameters

Parameters	Description
IN const char* pszOldPathName	Original name and path of the file.
IN const char* pszNewPathName	New name and path of the file.

Returns

- true: File renamed successfully.
- false: Cannot rename file. It may be already opened or does not exist.

Description

Renames the specified file. It will also move it to another directory if the path has been changed.

The use of this method is deprecated in favor of the new method CFileTools::Rename (see page 480)().

See Also

Kernel/CFileTools.h

2.10.3.4 - MxTraceCallOutputHandler Function

Calls the output handler.

C++

```
unsigned int MxTraceCallOutputHandler(unsigned int uNumberOfMessages);
```

Parameters

Parameters	Description
unsigned int uNumberOfMessages	Number of messages that need to be outputted. If this is 0 the memory queue will be emptied.

Returns

Number of messages that were outputted. This will be smaller then the uNumberOfMessages if the queue is empty and equal if there are still messages in the memory queue.

Description

Calls all the output handlers in order to output traces stored in the memory queue. For performance reasons, this function is not protected against concurrent calls from different threads. It must not be called after the CFrameworkInitializer::Finalize (see page 790) call. This function is only available if MXD_TRACE_USE_EXTERNAL_THREAD (see page 314) is defined.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.10.3.5 - MxTraceSetThreadMessagesPerPeriod Function

Sets the internal thread number of messages outputted per period.

C++

```
void MxTraceSetThreadMessagesPerPeriod(unsigned int uTraceMessagesPerPeriod);
```

Parameters

Parameters	Description
unsigned int uTraceMessagesPerPeriod	Number of messages outputted per period.

Description

Sets the number of messages the internal thread will output per period.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.10.3.6 - MxTraceSetThreadPeriodMs Function

Sets the internal thread waiting period.

C++

```
void MxTraceSetThreadPeriodMs(unsigned int uTracePeriodMs);
```

Parameters

Parameters	Description
unsigned int uTracePeriodMs	Waiting period between trace outputting.

Description

Sets the period the internal thread will sleep between outputting.

Location

Defined in Basic/MxTrace.h but must include Config/MxConfig.h to access this.

2.10.4 - Macros

This section documents the macros of the Sources/Kernel folder.

Macros

Macro	Description
MX_DELETE (see page 509)	Deletes an object.
MX_DELETE_ARRAY (see page 510)	Deletes an array of objects.
MX_NEW (see page 510)	Creates an object.
MX_NEW_ARRAY (see page 510)	Creates an array of objects.

2.10.4.1 - MX_DELETE Macro

Deletes an object.

C++

```
#define MX_DELETE(pInstance)
```

Parameters

Parameters	Description
pInstance	Pointer to the object to delete. If NULL, no operation is performed.

Description

This macro deletes an object created using MX_NEW (see page 510). This macro replaces the delete operator and should be used even if the memory allocator is not enabled.

Two implementations of the macro are provided. When MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is defined, the macro uses the memory allocator. Otherwise, it uses the global delete operator. The utilisation of the memory allocator centralizes the memory management process. This gives us the ability to gather statistics, to perform bound checking and memory leak detection.

Note that the memory allocator bypasses the protected and private access controls of the classes' destructor. Therefore, it is possible to call a non-public destructor using the MX_DELETE macro when MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is defined.

IMPORTANT: This lack of control MUST not be used in any way. It is STRICTLY PROHIBITED to implement code that will call MX_DELETE on a pointer that has a protected or private destructor. Furthermore, the code will not compile when MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is not defined.

Location

Defined in "CMemoryAllocator.h".

See Also

MX_DELETE_ARRAY (see page 510), MX_NEW (see page 510), MX_NEW_ARRAY (see page 510),
MxD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

2.10.4.2 - MX_DELETE_ARRAY Macro

Deletes an array of objects.

C++

```
#define MX_DELETE_ARRAY(pArray)
```

Parameters

Parameters	Description
pArray	A pointer to the first object of the array.

Description

This macro deletes an array of objects that was created using the MX_NEW_ARRAY (see page 510) macro. This macro replaces the delete[] operator and should be used even if the memory allocator is not enabled.

Two implementations of the macro are provided. When MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is defined, the macro uses the memory allocator. Otherwise, it uses the global new operator. The utilisation of the memory allocator centralizes the memory management process. This gives us the ability to gather statistics, to perform bound checking and memory leak detection.

Location

Defined in "CMemoryAllocator.h".

See Also

MX_NEW (see page 510), MX_DELETE (see page 509), MX_NEW_ARRAY (see page 510), MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

2.10.4.3 - MX_NEW Macro

Creates an object.

C++

```
#define MX_NEW(varType)
```

Parameters

Parameters	Description
varType	Type of the object to create.

Returns

A pointer to the created object.

Description

This macro creates an object of type varType. That object must later be deleted with MX_DELETE (see page 509). This macro replaces the new operator and should be used even if the memory allocator is not enabled.

Two implementations of the macro are provided. When MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is defined, the macro uses the memory allocator. Otherwise, it uses the global delete operator. The utilisation of the memory allocator centralizes the memory management process. This gives us the ability to gather statistics, to perform bound checking and memory leak detection.

Warning

For templated classes with multiple arguments (i.e. CMap (see page 186)), typedef MUST be used in order to use MX_NEW with these classes. Otherwise, compilation errors will occur.

Location

Defined in "CMemoryAllocator.h".

See Also

MX_DELETE_ARRAY (see page 510), MX_DELETE (see page 509), MX_NEW_ARRAY (see page 510), MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

2.10.4.4 - MX_NEW_ARRAY Macro

Creates an array of objects.

C++

```
#define MX_NEW_ARRAY(varType, uSize)
```

Parameters

Parameters	Description
varType	Type of the objects to create.
uSize	Size of the array to create

Returns

A pointer to the first object of the created array of objects.

Description

This macro creates an array of uSize objects of type varType. That array must later be deleted with MX_DELETE_ARRAY (see page 510). This macro replaces the new[] operator and should be used even if the memory allocator is not enabled.

Two implementations of the macro are provided. When MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296) is defined, the macro uses the memory allocator. Otherwise, it uses the global new operator. The utilisation of the memory allocator centralizes the memory management process. This gives us the ability to gather statistics, to perform bound checking and memory leak detection.

Location

Defined in "CMemoryAllocator.h".

See Also

MX_DELETE_ARRAY (see page 510), MX_DELETE (see page 509), MX_NEW (see page 510),
MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT (see page 296).

2.10.5 - Structures

This section documents the structures of the Sources/Kernel folder.

Structs

Struct	Description
SStat (see page 511)	Structure holding the information related to a file. Not all data members will be filled on all operating systems.
SMemoryBlockExtraInfo (see page 512)	Structure used to store extra information about an allocated memory block.
SMemoryBlockHeader (see page 513)	Structure used to store information about an allocated memory block.
SMemoryPoolInfo (see page 514)	
SMemoryStatistics (see page 514)	Structure to store memory statistics.
SThreadStackInfo (see page 514)	Structure to store stack usage information.

2.10.5.1 - CFile::SStat Struct

Structure holding the information related to a file. Not all data members will be filled on all operating systems.

C++

```
struct SStat {
    uint32_t uDevice;
    uint32_t uInode;
    uint16_t uMode;
    int16_t nLinkCount;
    int16_t nUserId;
    int16_t nGroupId;
    uint32_t uDeviceType;
    int32_t nSize;
    time_t nAccessTime;
    time_t nModificationTime;
    time_t nChangeTime;
    int32_t nBlockSize;
    int32_t nBlockCount;
    uint8_t uAttribute;
};
```

Members

Members	Description
uint32_t uDevice;	Device of the file.

uint32_t uInode;	Inode of the file.
uint16_t uMode;	Mode of the file.
int16_t nLinkCount;	Link count of the file.
int16_t nUserId;	User ID of the file.
int16_t nGroupId;	Group ID of the file.
uint32_t uDeviceType;	Device type of the file.
int32_t nSize;	Size of the file in bytes.
time_t nAccessTime;	Access time of the file.
time_t nModificationTime;	Modification time of the file.
time_t nChangeTime;	Change time of the file.
int32_t nBlockSize;	Block size of the file.
int32_t nBlockCount;	Block count of the file.
uint8_t uAttribute;	Attributes of the file.

2.10.5.2 - CMemoryAllocator::SMemoryBlockExtraInfo Struct

Structure used to store extra information about an allocated memory block.

Class Hierarchy

C++

```
struct SMemoryBlockExtraInfo {
    char const* m_pszType;
    char const* m_pszFilename;
    enum EMemoryBlockFlags {
        eMEMORY_BLOCK_TEMPORARILY_UNTRACKED_FLAG = 0x0001,
        eMEMORY_BLOCK_POOLED = 0x0002
    };
    uint16_t m_uFlags;
    uint16_t m_uLineNumber;
};
```

Description

This structure holds extra information about an allocated memory block. This information is not crucial for the memory allocation mechanism to function, it can however be very useful for debugging purposes. The information includes the file and line at which the allocation was done and the type of object allocated.

See Also

SMemoryBlockHeader (see page 513)

Constructors

Constructor	Description
SMemoryBlockExtraInfo (see page 513)	Constructor for inline construction of a SMemoryBlockInfo.

Legend



2.10.5.2.1 - Data Members

2.10.5.2.1.1 - CMemoryAllocator::SMemoryBlockExtraInfo::m_pszFilename Data Member

```
char const* m_pszFilename;
```

Description

The filename of the file in which the memory allocation was done.

2.10.5.2.1.2 - CMemoryAllocator::SMemoryBlockExtraInfo::m_pszType Data Member

```
char const* m_pszType;
```

Description

A string representing the Type of data contained in the memory block.

2.10.5.2.1.3 - CMemoryAllocator::SMemoryBlockExtraInfo::m_uFlags Data Member

```
uint16_t m_uFlags;
```

Description

Bitfield of flags used to collect various information about the memory block.

2.10.5.2.1.4 - CMemoryAllocator::SMemoryBlockExtraInfo::m_uLineNumber Data Member

```
uint16_t m_uLineNumber;
```

Description

The line number in the file at which the memory allocation was done.

2.10.5.2.2 - Constructors

2.10.5.2.2.1 - CMemoryAllocator::SMemoryBlockExtraInfo::SMemoryBlockExtraInfo Constructor

Constructor for inline construction of a SMemoryBlockInfo.

C++

```
inline explicit SMemoryBlockExtraInfo(IN char const* szType, IN char const* szFilename, IN uint16_t uFlags, IN
uint32_t m_uLineNumber);
```

Parameters

Parameters	Description
IN char const* szType	A null-terminated string representing the type of object allocated.
IN char const* szFilename	A null-terminated string containing the filename of the file in which the allocation was done.
IN uint16_t uFlags	Flags to be set in the SMemoryBlockExtraInfo.
uLineNumber	An integer representing the line number at which the allocation was done.

Description

Inline Constructor for an SMemoryBlockInfo structure. This constructor allows the complete initialization of a SMemoryBlockInfo structure to allow ease of use in a Macro.

2.10.5.3 - CMemoryAllocator::SMemoryBlockHeader Struct

Structure used to store information about an allocated memory block.

C++

```
struct SMemoryBlockHeader {
    SMemoryBlockExtraInfo m_stExtraInformation;
    size_t m_uBlockSize;
    uint32_t m_uHighBlockStartSignature;
    uint32_t m_uLowBlockStartSignature;
};
```

Description

This structure holds information about an allocated memory block. The information held in this header includes crucial memory management information as well as optionally including extra debugging information.

See Also

SMemoryBlockExtraInfo (see page 512), SMemoryBlock

Members

Members	Description
SMemoryBlockExtraInfo m_stExtraInformation;	Extra information about the memory block.
size_t m_uBlockSize;	The size of the block.
uint32_t m_uHighBlockStartSignature;	The high double word of the start-of-block signature.
uint32_t m_uLowBlockStartSignature;	The low double word of the start-of-block signature.

2.10.5.4 - CMemoryAllocator::SMemoryPoolInfo Struct

```
struct SMemoryPoolInfo {
    size_t m_uBlockSizeLowerBound;
    size_t m_uBlockSizeUpperBound;
    size_t m_uCapacity;
    PFNAllocateFunction m_pfnAllocate;
    PFNDeallocateFunction m_pfnDeallocate;
};
```

Description

This structure describes the parameters of a memory pool. A memory pool is an entity which can store blocks that are no longer used for later use when requested by the program.

To enable memory pooling you need to define the MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT (see page 296) compiler flag and configure the memory pooling using the MXD_astMEMORY_ALLOCATOR_POOL_INFO (see page 274) compiler flag according to the specific format described with its documentation.

See Also

MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT (see page 296)

MXD_astMEMORY_ALLOCATOR_POOL_INFO (see page 274) CPooledMemoryBlockAllocator

Members

Members	Description
size_t m_uBlockSizeLowerBound;	The lower bound size that the memory pool can accept (exclusive).
size_t m_uBlockSizeUpperBound;	The upper bound size that the memory pool can accept (inclusive).
size_t m_uCapacity;	The capacity of the pool.
PFNAllocateFunction m_pfnAllocate;	A pointer to the allocation function be used with this pool.
PFNDeallocateFunction m_pfnDeallocate;	A pointer to the deallocation function to be used with this pool.

2.10.5.5 - CMemoryAllocator::SMemoryStatistics Struct

Structure to store memory statistics.

C++

```
struct SMemoryStatistics {
    uint32_t m_uAllocatedMemory;
    unsigned int m_uNewCallCount;
    unsigned int m_uDeleteCallCount;
    uint32_t m_uBiggestMemoryBlock;
    uint32_t m_uPeakMemoryUsage;
    uint32_t m_uHighWaterMark;
    uint32_t m_uLowWaterMark;
    unsigned int m_auMemoryBlocksCount[uMEMORY_BLOCKS_INTERVALS];
};
```

Description

This structure is used to store memory statistics.

Members

Members	Description
uint32_t m_uAllocatedMemory;	A uint32_t (see page 85) value representing the total number of bytes allocated.
unsigned int m_uNewCallCount;	An unsigned int value representing the number of calls the new operator.
unsigned int m_uDeleteCallCount;	An unsigned int value representing the number of calls to the delete operator.
uint32_t m_uBiggestMemoryBlock;	A uint32_t (see page 85) value representing the size in bytes of the biggest allocated memory block.
uint32_t m_uPeakMemoryUsage;	A uint32_t (see page 85) value representing the peak memory usage in bytes.
uint32_t m_uHighWaterMark;	A uint32_t (see page 85) value representing the upper limit of allocated memory used to notify the eEN_CMEMORYALLOCATOR_TRACKING event observer.
uint32_t m_uLowWaterMark;	A uint32_t (see page 85) value representing the lower limit of allocated memory used to notify the eEN_CMEMORYALLOCATOR_TRACKING event observer.
unsigned int m_auMemoryBlocksCount[uMEMORY_BLOCKS_INTERVALS];	An array of unsigned int of size uMEMORY_BLOCKS_INTERVALS (see page 522). Each value in the array represents the number of allocated blocks whose size falls between $2^{\text{index}-1}$ and 2^{index} .

2.10.5.6 - CThread::SThreadStackInfo Struct

Structure to store stack usage information.

C++

```
struct SThreadStackInfo {
    unsigned int m_uPeakUsage;
    char m_szThreadName[g_uTHREAD_NAME_MAX_SIZE];
};
```

Description

This structure is used to store different information related to stack usage.

Members

Members	Description
unsigned int m_uPeakUsage;	The peak usage in the stack of the thread. Note that this value is only an approximate.
char m_szThreadName[g_uTHREAD_NAME_MAX_SIZE];	The name of the thread.

2.10.6 - Templates

This section documents the templates of the Sources/Kernel folder.

Templates

Template	Description
CAtomicOperations (see page 515)	Encapsulates atomic operation of any types.
CAtomicValue (see page 518)	Represents an atomic value on which atomic operations can be performed.

2.10.6.1 - CAtomicOperations Template

Encapsulates atomic operation of any types.

Class Hierarchy

CAtomicOperations

C++

```
template <class _Type>
class CAtomicOperations;
```

Description

CAtomicOperations is a templated class which encapsulates atomic operations on any type. In its generic implementation, atomic operations are enforced using a mutex. Templated specialization of this class can be done for any platform that natively supports atomic operations. It is currently implemented natively for the Windows, Linux and Linux PPC platforms only.

This class is intended to be used directly in cases where native atomic operations are not supported and the memory footprint of the code is a constraint. By using this class, instead of the safer CAtomicValue (see page 518) you can avoid having one mutex per synchronized value, and synchronize multiple values with one mutex. In other cases, it is safer to use the CAtomicValue (see page 518) class.

Template specialization of the class can be done for every platform natively supporting atomic operations.

Current Native Support (OS / Compiler / Architecture):

- Windows / MSVC / x86
 - int16_t (see page 85)
 - uint16_t (see page 85)
 - int32_t (see page 85)
 - uint32_t (see page 85)
- Linux / GCC / x86
 - int16_t (see page 85)
 - uint16_t (see page 85)
 - int32_t (see page 85)
 - uint32_t (see page 85)
- Linux / GCC / PPC
 - int32_t (see page 85)

- `uint32_t` (see page 85)

Location

Kernel/CAtomic.h

See Also

`CAtomicValue` (see page 518)

Constructors

Constructor	Description
<code>~CAtomicOperations</code> (see page 516)	Default Constructor.

Legend



Destructors

Destructor	Description
<code>~CAtomicOperations</code> (see page 517)	Destructor.

Legend



Methods

Method	Description
<code>Decrement</code> (see page 517)	Decrements the value in an atomic manner and returns it.
<code>Exchange</code> (see page 517)	Exchanges the value in an atomic manner and returns the value before the operation was executed.
<code>Increment</code> (see page 517)	Increments the value in an atomic manner and returns it.
<code>Retrieve</code> (see page 518)	Retrieves the value in an atomic manner.

Legend



2.10.6.1.1 - Constructors

2.10.6.1.1.1 - `CAtomicOperations`

2.10.6.1.1.1.1 - `CAtomicOperations::CAtomicOperations` Constructor

Default Constructor.

C++

```
CAtomicOperations();
```

Description

Default constructor for the `CAtomicOperations` class that creates a mutex owned by this instance of the `CAtomicOperations` class.

2.10.6.1.1.1.2 - `CAtomicOperations::CAtomicOperations` Constructor

Specialized constructor used to pass in a shared mutex to use for the atomic operations.

C++

```
CAtomicOperations(IN CMutex* pmutexShared);
```

Parameters

Parameters	Description
<code>IN CMutex* pmutexShared</code>	Parameter that allows specification of a shared mutex to be used for synchronizing the atomic operations. Note that if a mutex is specified, it is not owned by the <code>CAtomicOperations</code> class and needs to be valid for the lifetime of the <code>CAtomicOperations</code> instance.

Description

Specialized constructor for the CAtomicOperations class that allows specification of a shared mutex that will be used to synchronize the atomic operations.

2.10.6.1.2 - Destructors**2.10.6.1.2.1 - CAtomicOperations::~CAtomicOperations Destructor**

Destructor.

C++

```
~CAtomicOperations();
```

Description

Destructor for the CAtomicOperations (see page 515) class that destroys the mutex if it is owned by this instance of the CAtomicOperations (see page 515) class.

2.10.6.1.3 - Methods**2.10.6.1.3.1 - CAtomicOperations::Decrement Method**

Decrements the value in an atomic manner and returns it.

C++

```
_Type Decrement(INOUT _Type* pValue) const;
```

Parameters

Parameters	Description
INOUT _Type* pValue	A pointer to the value that must atomically decremented.

Returns

The decremented value.

Description

Method that performs an atomic decrement on the value pointed to by the passed pointer.

2.10.6.1.3.2 - CAtomicOperations::Exchange Method

Exchanges the value in an atomic manner and returns the value before the operation was executed.

C++

```
_Type Exchange(IN _Type source, INOUT _Type* pDestination) const;
```

Parameters

Parameters	Description
IN _Type source	The new value to be exchanged with the value pointed to by the pDestination parameter.
INOUT _Type* pDestination	Pointer to the value to be exchanged with the source.

Returns

The value of pDestination before the atomic exchange operation is done.

Description

Method that atomically exchanges the existing value pointed to by the specified destination with the new value specified in the source.

2.10.6.1.3.3 - CAtomicOperations::Increment Method

Increments the value in an atomic manner and returns it.

C++

```
_Type Increment(INOUT _Type* pValue) const;
```

Parameters

Parameters	Description
INOUT _Type* pValue	A pointer to the value that must be atomically incremented.

Returns

The incremented value.

Description

Method that performs an atomic increment on the value pointed to by the passed pointer.

2.10.6.1.3.4 - CAtomicOperations::Retrieve Method

Retrieves the value in an atomic manner.

C++

```
_Type Retrieve(INOUT _Type const* pValue) const;
```

Parameters

Parameters	Description
INOUT _Type const* pValue	A pointer to the value that should atomically retrieved.

Returns

The retrieved value.

Description

Method that atomically retrieves the value pointed to by the passed pointer.

2.10.6.2 - CAtomicValue Template

Represents an atomic value on which atomic operations can be performed.

Class Hierarchy**C++**

```
template <class _Type>
class CAtomicValue : private CAtomicOperations<_Type>;
```

Description

Class representing an atomic value on which multiple operations can be performed. The class derives from CAtomicOperations (see page 515) to provide the atomic functionality.

Notes

CAtomicValue derives from CAtomicOperations (see page 515) instead of having a CAtomicOperations (see page 515) member so that if a specialized templated version of CAtomicOperations (see page 515) that doesn't contain any members is used, CAtomicValue will not gain one byte in size for having an empty class member.

Location

Kernel/CAtomic.h CAtomicOperations

See Also

CMutex (see page 488)

Example

```
class RefCountedClass
{
    RefCountedClass()
        : m_atomicRefCount(1)
    {
    }
```

```

    uint32_t AddRef()
    {
        return m_atomicRefCount.Increment();
    }

    uint32_t Release()
    {
        uint32_t uResult = m_atomicRefCount.Decrement();
        if (uResult < 1)
        {
            delete this;
        }
        return result;
    }

protected:
    virtual ~RefCountedClass()
    {
        MX_ASSERT(m_atomicRefCount.Get() == 0);
    }

private:
    CAtomicValue<uint32_t> m_atomicRefCount;
};

```

Constructors

Constructor	Description
CAAtomicValue (see page 520)	Default constructor.

CAAtomicOperations Template

CAAtomicOperations Template	Description
CAAtomicOperations (see page 516)	Default Constructor.

Legend

	Constructor
	Method

Destructors

CAAtomicOperations Template

CAAtomicOperations Template	Description
~CAAtomicOperations (see page 517)	Destructor.

Legend

	Destructor
	Method

Methods

Method	Description
Decrement (see page 520)	Decrements the value in an atomic manner and returns it.
Exchange (see page 520)	Exchanges the value in an atomic manner and returns the old value before the operation was executed.
Increment (see page 521)	Increments the value in an atomic manner and returns it.
Retrieve (see page 521)	Retrieves the value in an atomic manner.

CAAtomicOperations Template

CAAtomicOperations Template	Description
Decrement (see page 517)	Decrements the value in an atomic manner and returns it.
Exchange (see page 517)	Exchanges the value in an atomic manner and returns the value before the operation was executed.
Increment (see page 517)	Increments the value in an atomic manner and returns it.
Retrieve (see page 518)	Retrieves the value in an atomic manner.

Legend

	Method
	Constructor

2.10.6.2.1 - Constructors

2.10.6.2.1.1 - CAtomicValue

2.10.6.2.1.1.1 - CAtomicValue::CAtomicValue Constructor

Default constructor.

C++

```
CAtomicValue();
```

Description

Default constructor for the CAtomicValue class. The default constructor will initialize the value to the default constructed `_Type` type.

2.10.6.2.1.1.2 - CAtomicValue::CAtomicValue Constructor

Specialized constructor that initializes the constructed CAtomicValue to the specified value.

C++

```
CAtomicValue(_Type const& initialValue);
```

Parameters

Parameters	Description
<code>_Type const& initialValue</code>	The initial value.

Description

Specialized constructor that allows the specification of an initial value.

2.10.6.2.2 - Methods

2.10.6.2.2.1 - CAtomicValue::Decrement Method

Decrements the value in an atomic manner and returns it.

C++

```
_Type Decrement();
```

Returns

The decremented value of this CAtomicValue (see page 518) object.

Description

Method that performs an atomic decrement.

2.10.6.2.2.2 - CAtomicValue::Exchange Method

Exchanges the value in an atomic manner and returns the old value before the operation was executed.

C++

```
_Type Exchange(_Type const& newValue = _Type());
```

Parameters

Parameters	Description
<code>_Type const& newValue = _Type()</code>	The new value.

Returns

The original value of this CAtomicValue (see page 518) object.

Description

Method that atomically exchanges the existing value with a new one.

2.10.6.2.2.3 - CAtomicValue::Increment Method

Increments the value in an atomic manner and returns it.

C++

```
_Type Increment();
```

Returns

The incremented value of this CAtomicValue (see page 518) object.

Description

Method that performs an atomic increment.

2.10.6.2.2.4 - CAtomicValue::Retrieve Method

Retrieves the value in an atomic manner.

C++

```
_Type Retrieve() const;
```

Returns

The current value of this CAtomicValue (see page 518) object.

Description

Method that atomically retrieves the internal value of this CAtomicValue (see page 518) object. Note that this value simply represents a snapshot of this CAtomicValue (see page 518) in its lifetime. It is dangerous to use the Retrieve method to perform synchronization as it will lead to race conditions.

2.10.7 - Types

This section documents the types of the Sources/Kernel folder.

Types

Type	Description
mxt_fd (see page 521)	File descriptor basic data type.

2.10.7.1 - mxt_fd Type

File descriptor basic data type.

C++

```
typedef int mxt_fd;
```

Description

The file descriptor is used to access a file. It is created when opening a file and is usually valid until the file is closed. All methods of CFile (see page 472) use this descriptor when calling the operating system's file primitives.

On all operating systems, the descriptor is an integer except on Symbian where it is a pointer to a RFile object.

A file descriptor should never be kept because it can be invalidated by the following methods: Close, Truncate. On Windows CE, the Stat methods can also invalidate a file descriptor.

See Also

GetFileDescriptor

2.10.8 - Variables

This section documents the variables of the Sources/Kernel folder.

2.10.8.1 - g_uMEMORY_BLOCK_OVERHEAD_SIZE Variable

```
const size_t g_uMEMORY_BLOCK_OVERHEAD_SIZE = 0;
```

Description

Overhead size of each memory block. There is no memory block overhead when the memory allocator is not enabled.

2.10.8.2 - g_uTHREAD_NAME_MAX_SIZE Variable

```
const unsigned int g_uTHREAD_NAME_MAX_SIZE = 32;
```

Description

Default size of thread name (32 bytes, including the NULL character).

2.10.8.3 - uMEMORY_BLOCKS_INTERVALS Variable

```
const unsigned int uMEMORY_BLOCKS_INTERVALS = 32;
```

Description

Set interval between memory blocks.

2.11 - Network

This section documents the Sources/Network folder of the M5T Framework. It is divided in functional subsections:

- Classes ([see page 522](#))
- Enumerations ([see page 650](#))
- Functions ([see page 652](#))
- Structures ([see page 662](#))
- Variables ([see page 663](#))

2.11.1 - Classes

This section documents the classes of the Sources/Network folder.

Classes

Class	Description
CAsyncSocketFactory (see page 523)	This class centralizes the creation of asynchronous sockets.
CFqdn (see page 527)	This class is a simple container for FQDN/port number association.
CMac (see page 531)	This class stores and manages the Ethernet MAC address.
CPollRequestStatus (see page 536)	Class providing polled events detection on multiple TRequestStatus.
CPollSocket (see page 540)	Class providing polled events detection on multiple sockets.
CSocket (see page 544)	Base socket class.
CSocketAddr (see page 545)	Class providing a common interface for storing socket addresses.
CTcpServerSocket (see page 559)	Implements a TCP socket in servermode.
CTcpSocket (see page 567)	Implementation of a client TCP socket.
CTcpSocketOptions (see page 579)	An ECOM (see page 412) object that can be used to configure TCP socket options.
CUdpSocket (see page 585)	Implementation of a client UDP socket.
CUdpTracing (see page 597)	UDP trace output handler.
IAsyncClientSocket (see page 599)	Interface defining the methods associated with a client socket.
IAsyncClientSocketMgr (see page 601)	Interface through which asynchronous client sockets report their events.
IAsyncIOSocket (see page 602)	Interface defining the methods accessible on connected asynchronous sockets to send and receive data.
IAsyncIOSocketMgr (see page 605)	This is the interface through which the asynchronous I/O sockets report their events.
IAsyncServerSocket (see page 607)	The IServerSocket (see page 644) interface defines the methods needed for a server type of socket.
IAsyncServerSocketMgr (see page 610)	This is the interface through which server sockets report events.
IAsyncSocket (see page 612)	Interface defining the basic methods accessible on asynchronous sockets. Class implementing abstraction of sockets. It is used to encapsulate the socket functionality in an asynchronous manner.
IAsyncSocketBufferSizeOptions (see page 620)	Interface defining the socket options that are related to buffer size.
IAsyncSocketFactoryConfigurationMgr (see page 621)	This interface is used to receive asynchronous sockets configuration requests from the socket factory.
IAsyncSocketFactoryCreationMgr (see page 622)	This interface is used to receive asynchronous sockets creation requests from the socket factory.

IAsyncSocketMgr (see page 624)	Defines basic events that can be reported by all types of asynchronous sockets.
IAsyncSocketQualityOfServiceOptions (see page 625)	Interface defining the socket options that are related to quality of service.
IAsyncSocketTcpOptions (see page 627)	Interface defining the options that are configurable on a TCP socket.
IAsyncSocketUdpOptions (see page 629)	Interface defining the options that are configurable on a UDP socket.
IAsyncSocketWindowsGqosOptions (see page 630)	Interface defining the socket options that are related to quality of service. This interface is specialized for the Windows GQOS flowspec option.
IAsyncUnconnectedIoSocket (see page 631)	Interface defining the methods accessible on unconnected asynchronous sockets to send and receive data.
IAsyncUnconnectedIoSocketMgr (see page 634)	This is the interface through which the unconnected asynchronous socket implementations report their events.
IClientSocket (see page 635)	Interface defining the methods associated with a client socket.
IoSocket (see page 637)	Interface defining the methods used by a socket to send and receive data.
IPollRequestStatusMgr (see page 642)	Interface used for CPollRequestStatus (see page 536).
IPollSocketMgr (see page 643)	Interface used for CPollSocket (see page 540).
IServerSocket (see page 644)	Interface defining the methods needed for a server socket.
ISocket (see page 646)	Interface offering common functionalities to all types of sockets.
ITcpSocketOptionsConfigure (see page 650)	Interface allowing the client to provide the necessary configuration information required by objects engaged in setting TCP socket options.

Functions

Function	Description
GetSockOptError (see page 599)	Gets the error status of the socket.

2.11.1.1 - CAsyncSocketFactory Class

This class centralizes the creation of asynchronous sockets.

Class Hierarchy

CAsyncSocketFactory

C++

```
class CAsyncSocketFactory;
```

Description

This class is an asynchronous socket factory that centralizes the creation of asynchronous sockets. The behaviour of the factory may be modified by the registration of managers. Two types of managers exist: creation managers and configuration managers.

The creation manager will be notified when an asynchronous socket must be created. The manager must then decide, based on the type of socket to be created, if it can handle the request. If it can, it creates the new asynchronous socket and returns it back to the factory. If not, it just indicates to the factory that it cannot fulfill the request. The factory will then try the same process on the next registered manager, if one is available. Creation managers are called in the reverse registration order. This is needed so that the last creation manager to be registered may override the creation process for a specific asynchronous socket type. If no managers are available or if none of them created the asynchronous socket, the socket factory will default to its own internal creation method which supports TCP and UDP asynchronous sockets.

The configuration manager will be notified when an asynchronous socket has just been created and must be configured with various socket options. In contrast with the creation managers, all the configuration managers will be notified and have the opportunity to configure the asynchronous socket. The configuration managers will be called in their registration ordering so that last registered configuration manager may override configurations previously performed by other managers.

Location

Network/CAsyncSocketFactory.h

See Also

IAsyncSocketFactoryConfigurationMgr (see page 621), IAsyncSocketFactoryCreationMgr (see page 622)

Methods

Method	Description
◆ CreateAsyncSocket (see page 524)	Creates an asynchronous socket.
◆ GetSocketList (see page 524)	Returns a copy of the list of sockets.
◆ IsAsyncSocketInList (see page 525)	Indicates if the asynchronous socket is already in the internal list.
◆ RegisterConfigurationMgr (see page 525)	Registers a configuration manager.
◆ RegisterCreationMgr (see page 525)	Registers a creation manager.
◆ RemoveSocketFromFactoryList (see page 526)	Removes a socket from the internal list.

• UnregisterConfigurationMgr (see page 526)	Unregisters a configuration manager.
• UnregisterCreationMgr (see page 527)	Unregisters a creation manager.

Legend

	Method
---	--------

2.11.1.1.1 - Methods

2.11.1.1.1.1 - CAAsyncSocketFactory::CreateAsyncSocket Method

Creates an asynchronous socket.

C++

```
static mxt_result CreateAsyncSocket(IN IEComUnknown* pServicingThread, IN const char* const* apszType, IN
unsigned int uTypeSize, OUT IAsyncSocket** ppAsyncSocket);
```

Parameters

Parameters	Description
IN IEComUnknown* pServicingThread	The servicing thread used to activate the asynchronous socket. If NULL is passed, the asynchronous socket automatically creates its own servicing thread with default parameters.
IN const char* const* apszType	An array of strings representing the type of the socket. It cannot be NULL and cannot contain a NULL item.
IN unsigned int uTypeSize	The size of the array of strings. It cannot be 0.
OUT IAsyncSocket** ppAsyncSocket	The location to where the new asynchronous socket will be returned. It cannot be NULL. The value of *ppAsyncSocket will be NULL if the creation fails.

Returns

resS_OK: The socket has been correctly created. resFE_FAIL: The socket creation failed or an error occurred during the socket configuration. resFE_INVALID_ARGUMENT: apszType contains at least one NULL item or ppAsyncSocket is NULL or uTypeSize is 0.

Description

This method creates an asynchronous socket. The type of the asynchronous socket to be created is provided as a sequence of network protocols starting from the highest level down to the lowest level followed possibly by arguments. When the socket is no longer useful, it must be released as per ECOM (see page 412) rules

Example

```
{
  "RTCP", "UDP" }, 2
  "RTP", "UDP" }, 2
  "RTP_RTCP", "UDP" }, 2
  "SIP", "TCP, m=client" }, 2
  "SIP", "TCP, m=server" }, 2
  "SIP", "TLS, m=client", "TCP" }, 3
  "SIP", "TLS, m=client", "TCP, m=client" }, 3
  "SIP", "TLS, m=server", "TCP" }, 3
  "SIP", "TLS, m=server", "TCP, m=server" }, 3
  "SIP", "UDP" }, 2
  "STUN", "TCP, m=client" }, 2
  "STUN", "TCP, m=server" }, 2
  "STUN", "TLS, m=client", "TCP" }, 3
  "STUN", "TLS, m=client", "TCP, m=client" }, 3
  "STUN", "TLS, m=server", "TCP" }, 3
  "STUN", "TLS, m=server", "TCP, m=server" }, 3
  "STUN", "UDP" }, 2
}
```

2.11.1.1.1.2 - CAAsyncSocketFactory::GetSocketList Method

Returns a copy of the list of sockets.

C++

```
static mxt_result GetSocketList(OUT CList<IAsyncSocket*>* plistOfSocket);
```

Parameters

Parameters	Description
plistOfSockets	A list of IAsyncSocket (see page 612) interface pointers to which the socket references will be copied. It cannot be NULL. The returned list can be empty.

Returns

resS_OK: The list has been copied. The returned list can be empty. resFE_FAIL: The plistOfSockets parameter is NULL.

Description

This method creates a copy of the internal socket list. An application that uses this method must make sure to release each socket reference contained in the returned list. This means that code like the following must be used when the list is released.

```
CList<IAsyncSocket*> list;
CAsyncSocketFactory::GetSocketList(OUT &list);

// Do something with the list.

// Release the list.
unsigned int uIndex = 0;
unsigned int uListSize = list.GetSize();
for(; uIndex < uListSize; uIndex++)
{
    list[uIndex]->ReleaseIfRef();
}
list.EraseAll();
```

2.11.1.1.3 - CAsyncSocketFactory::IsAsyncSocketInList Method

Indicates if the asynchronous socket is already in the internal list.

C++

```
static bool IsAsyncSocketInList(IN IAsyncSocket* pAsyncSocket);
```

Parameters

Parameters	Description
IN IAsyncSocket* pAsyncSocket	The pointer to the asynchronous socket to find in the internal list.

Returns

true: The socket is in the internal list. false: The socket is not in the internal list.

Description

Indicates if the asynchronous socket is already in the internal list.

2.11.1.1.4 - CAsyncSocketFactory::RegisterConfigurationMgr Method

Registers a configuration manager.

C++

```
static void RegisterConfigurationMgr(IN IAsyncSocketFactoryConfigurationMgr* pMgr);
```

Parameters

Parameters	Description
IN IAsyncSocketFactoryConfigurationMgr* pMgr	The manager to register.

Description

This method registers a configuration manager. The manager will be notified when an asynchronous socket has just been created and must be configured with various socket options like TOS (see page 39) byte and flow control. Duplicate managers are ignored.

2.11.1.1.5 - CAsyncSocketFactory::RegisterCreationMgr Method

Registers a creation manager.

C++

```
static void RegisterCreationMgr(IN IAsyncSocketFactoryCreationMgr* pMgr);
```

Parameters

Parameters	Description
IN IAsyncSocketFactoryCreationMgr* pMgr	The manager to register.

Description

This method registers a creation manager. The manager is called each time a new asynchronous socket needs to be created.

If no manager is registered or if no manager manages the creation of a specific asynchronous socket type, the socket factory will default to its own socket creation routine which supports TCP and UDP asynchronous sockets.

If multiple creation managers are registered, they will be called in LIFO order to make sure that the last registered manager will be called first and have priority over the creation process. Duplicate managers are ignored.

2.11.1.1.6 - CAsyncSocketFactory::RemoveSocketFromFactoryList Method

Removes a socket from the internal list.

C++

```
static mxt_result RemoveSocketFromFactoryList(IN IAsyncSocket* pAsyncSocket);
```

Parameters

Parameters	Description
IN IAsyncSocket* pAsyncSocket	The socket to remove from the list.

Returns

resSI_TRUE: The socket has been found and removed. resSI_FALSE: The socket has not been found. resFE_FAIL: The pAsyncSocket parameter is NULL.

Description

This method removes a socket from the list. It is primarily meant for internal use but an application implementing a custom type of sockets supported by the asynchronous socket factory will need to use it.

The custom socket should overload the CECComUnknown::NonDelegatingReleaseIfRef ECOM (see page 412) method in the following way:

```
unsigned int CSomeCustomSocketType::NonDelegatingReleaseIfRef()
{
    // Call base class for default behaviour.
    unsigned int uRefCount = CECComUnknown::NonDelegatingReleaseIfRef();

    // If the socket has been created through the socket factory, there will
    // always be 1 reference on the socket for the socket factory's list
    // reference. It must be removed for the socket to be completely released.
    // That reference should always be the last one.
    if (uRefCount == 1)
    {
        mxt_result res = resSI_FALSE;

        // Remove the reference in the socket factory's list.
        res = CAsyncSocketFactory::RemoveSocketFromFactoryList(this);

        // If the socket has not been created with the socket factory, it
        // will not be found in the socket list, and this means that the
        // application still has a real reference on the socket. Thus, the
        // real reference count must be returned.
        if (res == resSI_TRUE)
        {
            //That was the last reference. Let the caller know this fact.
            uRefCount = 0;
        }
    }

    // It is extremely important that no operations on the socket's data
    // member or method calls are done at this point. This is because the
    // call to RemoveSocketFromFactoryList will possibly re-enter this method
    // and the reference count will fall to 0. This in turn will call
    // MX_DELETE(this).
    //
    // So at this point in the method, it is possible that the 'this'
    // pointer is invalid.

    return uRefCount;
}
```

2.11.1.1.7 - CAsyncSocketFactory::UnregisterConfigurationMgr Method

Unregisters a configuration manager.

C++

```
static void UnregisterConfigurationMgr( IN IAsyncSocketFactoryConfigurationMgr* pMgr );
```

Parameters

Parameters	Description
IN IAsyncSocketFactoryConfigurationMgr* pMgr	The manager to unregister.

Description

This method unregisters a configuration manager.

2.11.1.1.1.8 - CAsyncSocketFactory::UnregisterCreationMgr Method

Unregisters a creation manager.

C++

```
static void UnregisterCreationMgr( IN IAsyncSocketFactoryCreationMgr* pMgr );
```

Parameters

Parameters	Description
IN IAsyncSocketFactoryCreationMgr* pMgr	The manager to unregister.

Description

This method unregisters a creation manager.

2.11.1.2 - CFqdn Class

This class is a simple container for FQDN/port number association.

Class Hierarchy
C++

```
class CFqdn;
```

Description

This class is a simple container for FQDN/port number association. The FQDN member is maintained as a CString (see page 126) while the port number is an integer. This class is here for reutilization purposes. It offers a very simple interface with a default/copy/initializer constructor. As well, it offers methods that simply provide access to data members using set and get. It also provides some basic operators like =, != and ==.

Location

Network/CFqdn.h

Constructors

Constructor	Description
CFqdn (see page 528)	Default Constructor.

Legend

Method

Destructors

Destructor	Description
~CFqdn (see page 529)	Destructor.

Legend

Method

Operators

Operator	Description
!= (see page 530)	Different than operator.
= (see page 530)	Assignment operator.

 == (see page 531)

Comparison operator.

Legend

	Method
---	--------

Methods

Method	Description
 GetFqdn (see page 529)	Gets the FQDN name.
 GetPort (see page 529)	Gets the FQDN port.
 Reset (see page 529)	Resets the FQDN.
 SetFqdn (see page 530)	Sets the FQDN name.
 SetPort (see page 530)	Sets the FQDN port.

Legend

	Method
---	--------

2.11.1.2.1 - Data Members

2.11.1.2.1.1 - CFqdn::m_strFqdn Data Member

CString (see page 126) containing the FQDN name.

C++

```
CString m_strFqdn;
```

2.11.1.2.1.2 - CFqdn::m_uPort Data Member

The FQDN port number.

C++

```
uint16_t m_uPort;
```

2.11.1.2.2 - Constructors

2.11.1.2.2.1 - CFqdn

2.11.1.2.2.1.1 - CFqdn::CFqdn Constructor

Default Constructor.

C++

```
CFqdn();
```

Description

Constructor

2.11.1.2.2.1.2 - CFqdn::CFqdn Constructor

Copy constructor.

C++

```
CFqdn(IN const CFqdn& rSrc);
```

Parameters

Parameters	Description
IN const CFqdn& rSrc	Reference to a CFqdn to copy.

Description

Copy constructor

2.11.1.2.2.1.3 - CFqdn::CFqdn Constructor

Constructor.

C++

```
CFqdn(IN const CString& rstrFqdn, IN const uint16_t uPort = 0);
```

Parameters

Parameters	Description
IN const CString& rstrFqdn	Reference to a CString containing the FQDN name.
IN const uint16_t uPort = 0	Port number of the FQDN.

Description

Constructor. Builds the fully qualified domain name using the name and port number provided.

2.11.1.2.3 - Destructors

2.11.1.2.3.1 - CFqdn::~CFqdn Destructor

Destructor.

C++

```
~CFqdn();
```

Description

Destructor

2.11.1.2.4 - Methods

2.11.1.2.4.1 - CFqdn::GetFqdn Method

Gets the FQDN name.

C++

```
CString GetFqdn() const;
```

Returns

The fully qualified domain name in string format.

Description

Returns the fully qualified domain name in string format.

2.11.1.2.4.2 - CFqdn::GetPort Method

Gets the FQDN port.

C++

```
uint16_t GetPort() const;
```

Returns

The fully qualified domain name's port number.

Description

Returns the fully qualified domain name's port number.

2.11.1.2.4.3 - CFqdn::Reset Method

Resets the FQDN.

C++

```
void Reset();
```

Description

Resets the data contained in the CFqdn (see page 527).

2.11.1.2.4.4 - CFqdn::SetFqdn Method

Sets the FQDN name.

C++

```
void SetFqdn(IN const CString& rstrFqdn);
```

Parameters

Parameters	Description
IN const CString& rstrFqdn	A reference to a CString (see page 126) holding the fully qualified domain name to set.

Description

Sets the name of the fully qualified domain name to use.

2.11.1.2.4.5 - CFqdn::SetPort Method

Sets the FQDN port.

C++

```
void SetPort(IN const uint16_t uPort);
```

Parameters

Parameters	Description
IN const uint16_t uPort	The port number to set to this CFqdn (see page 527)

Description

Sets the port number of the fully qualified domain name to use.

2.11.1.2.5 - Operators**2.11.1.2.5.1 - CFqdn::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CFqdn& rSrc) const;
```

Parameters

Parameters	Description
IN const CFqdn& rSrc	A reference to the CFqdn (see page 527) to compare with.

Returns

True if both CFqdns are different, false otherwise.

Description

Compares the two CFqdns to see if they are different.

2.11.1.2.5.2 - CFqdn::= Operator

Assignment operator.

C++

```
CFqdn& operator =(IN const CFqdn& rSrc);
```

Parameters

Parameters	Description
IN const CFqdn& rSrc	A reference to the CFqdn (see page 527) to assign.

Returns

A reference to the assigned CFqdn (see page 527).

Description

Assigns the right hand CFqdn (see page 527) to the left hand one.

2.11.1.2.5.3 - CFqdn::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CFqdn& rSrc) const;
```

Parameters

Parameters	Description
IN const CFqdn& rSrc	A reference to the CFqdn (see page 527) to compare with.

Returns

True if both CFqdns are equal, false otherwise.

Description

Compares the two CFqdns to see if they are equal.

2.11.1.3 - CMac Class

This class stores and manages the Ethernet MAC address.

Class Hierarchy

```
CMac
```

C++

```
class CMac;
```

Description

```
class CMac
```

This class stores and manages the Ethernet MAC address.

Notes

The MAC address is stored in the file named MAC.cfg

Published Interface:

Constructors

Constructor	Description
CMac (see page 532)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CMac (see page 532)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
\neq != (see page 535)	Different than operator.
\neq = (see page 536)	Assignment operator.
\neq == (see page 536)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
\neq GetMac (see page 533)	Gets the MAC address in byte format.
\neq GetMacStr (see page 533)	Gets the MAC address in string format.
\neq IsValid (see page 534)	Checks if the MAC address stored in this object is valid.
\neq SetMac (see page 534)	Sets the MAC address.
\neq SetWithDefaultMac (see page 535)	Sets the default value to this CMac object.

Legend

	Method
--	--------

2.11.1.3.1 - Constructors

2.11.1.3.1.1 - CMac

2.11.1.3.1.1.1 - CMac::CMac Constructor

Constructor.

C++

```
CMac();
```

Description

Constructor.

2.11.1.3.1.1.2 - CMac::CMac Constructor

Copy constructor.

C++

```
CMac(const CMac& rhs);
```

Parameters

Parameters	Description
const CMac& rhs	Reference to the CMac to copy.

Description

Copy constructor.

2.11.1.3.2 - Destructors

2.11.1.3.2.1 - CMac::~CMac Destructor

Destructor.

C++

```
virtual ~CMac();
```

Description

Destructor.

2.11.1.3.3 - Methods

2.11.1.3.3.1 - GetMac

2.11.1.3.3.1.1 - CMac::GetMac Method

Gets the MAC address in byte format.

C++

```
const uint8_t* GetMac() const;
```

Returns

The MAC address in a byte array.

Description

Gets the MAC address.

2.11.1.3.3.1.2 - CMac::GetMac Method

Gets the MAC address.

C++

```
bool GetMac(OUT CString& rstrMac) const;
```

Parameters

Parameters	Description
OUT CString& rstrMac	A reference to a CString (see page 126) to contain the MAC address.

Returns

True if the returned MAC address is valid, false otherwise.

Description

Gets the MAC address.

2.11.1.3.3.1.3 - CMac::GetMac Method

Gets the MAC address.

C++

```
bool GetMac(OUT stMac& rMac) const;
```

Parameters

Parameters	Description
OUT stMac& rMac	A reference to a stMac structure to contain the MAC address.

Returns

True if the returned MAC address is valid, false otherwise.

Description

Gets the MAC address.

2.11.1.3.3.2 - CMac::GetMacStr Method

Gets the MAC address in string format.

C++

```
const CString& GetMacStr() const;
```

Returns

A CString (see page 126) containing the MAC address.

Description

Gets the MAC address.

2.11.1.3.3.3 - IsValid**2.11.1.3.3.3.1 - CMac::IsValid Method**

Checks if the MAC address stored in this object is valid.

C++

```
bool IsValid() const;
```

Returns

True if the MAC address stored is valid, false otherwise.

Description

Checks if the MAC address contained within this CMac (see page 531) object is valid.

2.11.1.3.3.3.2 - CMac::IsValid Method

Checks if the string holds a valid MAC address.

C++

```
bool IsValid(IN const CString& rstrMac) const;
```

Parameters

Parameters	Description
IN const CString& rstrMac	A reference to a CString (see page 126) containing the MAC address.

Returns

True if the MAC address is valid, false otherwise.

Description

Validates the MAC address.

2.11.1.3.3.3.3 - CMac::IsValid Method

Checks if the MAC address is valid.

C++

```
bool IsValid(IN const stMac& rMac) const;
```

Parameters

Parameters	Description
IN const stMac& rMac	A reference to a stMac structure containing the MAC address.

Returns

True if the MAC address is valid, false otherwise.

Description

Validates the MAC address.

2.11.1.3.3.4 - SetMac**2.11.1.3.3.4.1 - CMac::SetMac Method**

Sets the MAC address.

C++

```
bool SetMac(IN const CString& rstrMac);
```

Parameters

Parameters	Description
IN const CString& rstrMac	Reference to a CString (see page 126) holding the MAC.

Returns

True if the passed MAC address is valid, false otherwise.

Description

Sets the MAC address.

2.11.1.3.3.4.2 - CMac::SetMac Method

Sets the MAC address.

C++

```
bool SetMac(IN const stMac& rstMac);
```

Parameters

Parameters	Description
IN const stMac& rstMac	Reference to a struct holding the MAC address.

Returns

True if the passed MAC address is valid, false otherwise.

Description

Sets the MAC address.

2.11.1.3.3.4.3 - CMac::SetMac Method

Sets the MAC address.

C++

```
bool SetMac(IN const uint8_t* puMac);
```

Parameters

Parameters	Description
IN const uint8_t* puMac	Pointer to a byte array containing the MAC address.

Returns

True if the passed MAC address is valid, false otherwise.

Description

Sets the MAC address.

2.11.1.3.3.5 - CMac::SetWithDefaultMac Method

Sets the default value to this CMac (see page 531) object.

C++

```
void SetWithDefaultMac();
```

Description

Sets a default MAC value to the CMac (see page 531) object.

2.11.1.3.4 - Operators**2.11.1.3.4.1 - CMac::!= Operator**

Different than operator.

C++

```
bool operator !=(const CMac& rMac) const;
```

Parameters

Parameters	Description
const CMac& rMac	Reference to the CMac (see page 531) to compare with.

Returns

True if both CMac (see page 531) objects are different, false otherwise.

Description

Checks if both CMac (see page 531) objects are different.

2.11.1.3.4.2 - CMac::= Operator

Assignment operator.

C++

```
CMac& operator =(const CMac& rhs);
```

Parameters

Parameters	Description
const CMac& rhs	Reference to the CMac (see page 531) to assign.

Returns

Reference to the assigned CMac (see page 531).

Description

Assigns the right hand CMac (see page 531) to the left hand one.

2.11.1.3.4.3 - CMac::== Operator

Comparison operator.

C++

```
bool operator ==(const CMac& rMac) const;
```

Parameters

Parameters	Description
const CMac& rMac	Reference to the CMac (see page 531) to compare with.

Returns

True if both CMac (see page 531) objects are equal, false otherwise.

Description

Checks if both CMac (see page 531) objects are equal.

2.11.1.4 - CPollRequestStatus Class Symbian OS ONLY

Class providing polled events detection on multiple TRequestStatus.

Class Hierarchy

CPollRequestStatus

C++

```
class CPollRequestStatus;
```

Description

This class is a CPollSocket (see page 540) look-alike for Symbian.

This class provides support for polled request status completion detection on TRequestStatus and notification of these events to the

manager associated with the request status for which completion was detected.

A request status must be registered before CPollRequestStatus polls it for completion. This is done with a call to RegisterRequestStatus (see page 540). The request status, the manager and an optional opaque value must be provided.

Once registered, it is possible to enable the completion detection. This is done with a call to EnableCompletionDetection (see page 538). A request status MUST be set to KRequestPending before calling EnableCompletionDetection (see page 538).

It is also possible to disable the completion detection. This is done with a call to DisableCompletionDetection (see page 538).

A request status must be unregistered when events detection is no longer required. This is done with a call to UnregisterRequestStatus (see page 540).

The method Poll (see page 539) must be called to poll each registered request status. The class will go through its request status list identifying the ones where completion is detected. If completion is detected, the method Poll (see page 539) will call IPolledRequestStatusMgr::EvPolledRequestStatusEventDetected to notify the manager about the completion detection. It is possible to provide a timeout to the method Poll (see page 539). If the timeout is different than 0, then the Poll (see page 539) method will block until a completion is detected or the timeout is reached.

This class has no thread nor concurrency protection. It is the user's responsibility to manage that. It was decided to give this flexibility. It prevents the performance costs of entering/exiting critical sections systematically while there is a possibility that the user calls all CPollRequestStatus methods from the same thread.

The main reason for the presence of this class is to request status completion detection in the context of asynchronous sockets into the same thread.

For the polling mechanism to work, each request status MUST be used in the context of a single thread. It SHOULD only be used in conjunction with a CServicingThread (see page 771) object.

Location

Network/CPollRequestStatus.h

See Also

IPolledRequestStatusMgr (see page 642), CServicingThread (see page 771)

Constructors

Constructor	Description
~CPollRequestStatus (see page 538)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CPollRequestStatus (see page 538)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
DisableCompletionDetection (see page 538)	Disables the detection of completion for a request status.
EnableCompletionDetection (see page 538)	Enables the detection of completion for a request status.
GetCompletionDetectionState (see page 539)	Gets whether or not detection is enabled.
GetRegisteredRequestStatusCount (see page 539)	Returns the number of request status currently registered.
Poll (see page 539)	This method is time critical and must be called periodically to detect events on registered request status.
RegisterRequestStatus (see page 540)	Adds a request status / Polled request status manager association in the list of polled request status.
UnregisterRequestStatus (see page 540)	Removes a request status from the list of polled request status.

Legend

	Method
---	--------

2.11.1.4.1 - Constructors

2.11.1.4.1.1 - CPollRequestStatus::CPollRequestStatus Constructor

Constructor.

C++

```
CPollRequestStatus();
```

Description

Constructor.

2.11.1.4.2 - Destructors

2.11.1.4.2.1 - CPollRequestStatus::~CPollRequestStatus Destructor

Destructor.

C++

```
virtual ~CPollRequestStatus();
```

Description

Destructor.

2.11.1.4.3 - Methods

2.11.1.4.3.1 - CPollRequestStatus::DisableCompletionDetection Method

Disables the detection of completion for a request status.

C++

```
mxt_result DisableCompletionDetection(IN TRequestStatus* pRequestStatus);
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to disable completion detection.

Returns

resFE_INVALID_ARGUMENT

Description

Disables the detection of the completion of a request.

See Also

[EnableCompletionDetection](#) (see page 538)

2.11.1.4.3.2 - CPollRequestStatus::EnableCompletionDetection Method

Enables the detection of completion for a request status.

C++

```
mxt_result EnableCompletionDetection(IN TRequestStatus* pRequestStatus);
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to enable the detection.

Returns

resFE_INVALID_ARGUMENT

Description

Enables the detection of the completion of a request. pRequestStatus MUST be set to KRequestPending before calling

EnableCompletionDetection.

See Also

DisableCompletionDetection (see page 538)

2.11.1.4.3.3 - CPollRequestStatus::GetCompletionDetectionState Method

Gets whether or not detection is enabled.

C++

```
mxt_result GetCompletionDetectionState(IN TRequestStatus* pRequestStatus, OUT bool& rbIsEnabled);
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status from which to get the detection state.
OUT bool& rbIsEnabled	On exit, will contain whether or not the detection is enabled.

Returns

resFE_INVALID_ARGUMENT

Description

Gets whether or not the completion detection was enabled for this request status.

See Also

EnableCompletionDetection (see page 538), DisableCompletionDetection (see page 538)

2.11.1.4.3.4 - CPollRequestStatus::GetRegisteredRequestStatusCount Method

Returns the number of request status currently registered.

C++

```
unsigned int GetRegisteredRequestStatusCount();
```

Returns

The number of registered request status

Description

Gets the number of registered request status in the list.

2.11.1.4.3.5 - CPollRequestStatus::Poll Method

This method is time critical and must be called periodically to detect events on registered request status.

C++

```
mxt_result Poll(IN uint64_t uTimeoutMs = 0);
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs = 0	The maximum time the Poll method is allowed to block.

Returns

resSI_TRUE is at least one request has completed.

Description

The Poll() method polls each registered request status to detect completion. If completion is detected, the associated manager is notified.

See Also

IPollRequestStatusMgr::EvPolledRequestStatusMgrEventDetected (see page 643)

2.11.1.4.3.6 - CPollRequestStatus::RegisterRequestStatus Method

Adds a request status / Polled request status manager association in the list of polled request status.

C++

```
mxt_result RegisterRequestStatus(IN TRequestStatus* pRequestStatus, IN IPolledRequestStatusMgr* pPolledRequestStatusMgr, IN mxt_opaque opq = MX_INT32_TO_OPQ(0));
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be registered.
IN IPolledRequestStatusMgr* pPolledRequestStatusMgr	The manager that will be called when a request completes.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value that will be provided to the manager.

Returns

resFE_INVALID_ARGUMENT

Description

Registers a request status for completion detection.

See Also

UnregisterRequestStatus (see page 540)

2.11.1.4.3.7 - CPollRequestStatus::UnregisterRequestStatus Method

Removes a request status from the list of polled request status.

C++

```
mxt_result UnregisterRequestStatus(IN TRequestStatus* pRequestStatus, OUT mxt_opaque* popq = NULL);
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be unregistered.
OUT mxt_opaque* popq = NULL	On exit, will contain the opaque value associated with the request status. May be NULL if the value should just be discarded.

Returns

resFE_INVALID_ARGUMENT

Description

Unregister a request status.

See Also

RegisterRequestStatus (see page 540)

2.11.1.5 - CPollSocket Class

Class providing polled events detection on multiple sockets.

Class Hierarchy

```
CPollSocket
```

C++

```
class CPollSocket;
```

Description

This class provides support for polled events detection on multiple sockets and notification of these events to the manager associated with the socket on which the event was detected. Three different events may be detected: readability, writability and in exception.

A socket must be registered before CPollSocket polls it for new events. This is done with a call to RegisterSocket (see page 544). The socket handle, the manager and an optional opaque value must be provided.

Once registered, it is possible to enable the detection of specific events. This is done with a call to EnableEventDetection.

It is also possible to disable the detection of specific events. This is done with a call to `DisableEventDetection`.

A socket must be unregistered when events detection is no longer required. This is done with a call to `UnregisterSocket` (see page 544). The socket handle and the event to unregister must be supplied.

The method `Poll` (see page 543) must be called to poll each registered socket. The class will go through its sockets identifying socket handles where events are available. If an event is detected for a socket, the method `Poll` (see page 543) will call `IPolledSocketMgr::EvPolledSocketMgrEventDetected` (see page 644) to notify the manager about the new events that have been detected. It is possible to pass a timeout to the method `Poll` (see page 543). If the timeout is different than 0, then the `Poll` (see page 543) method will block until an event is detected or the timeout is reached.

This class has no thread nor concurrency protection. It is the user's responsibility to manage that. It was decided to give this flexibility. It prevents the performance costs of entering/exiting critical sections systematically while there is a possibility that the user calls all `CPollSocket` methods from the same thread.

The main reason for the presence of this class is to regroup network reception for a group of sockets into the same thread instead of creating sockets all having their own thread.

Location

Network/CPollSocket

See Also

`IPolledSocketMgr` (see page 643)

Constructors

Constructor	Description
 <code>CPollSocket</code> (see page 541)	Default Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 <code>~CPollSocket</code> (see page 542)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 <code>DisableEventsDetection</code> (see page 542)	Disables the detection of specific events for a socket.
 <code>EnableEventsDetection</code> (see page 542)	Enables the detection of specific events for a socket.
 <code>GetEventsDetectionState</code> (see page 543)	Gets the events that are currently being detected.
 <code>GetRegisteredSocketCount</code> (see page 543)	Returns the number of sockets currently registered.
 <code>Poll</code> (see page 543)	This method is time critical and must be called periodically to detect events on registered sockets.
 <code>RegisterSocket</code> (see page 544)	Adds a socket handle / Polled socket manager association in the list of polled sockets.
 <code>UnregisterSocket</code> (see page 544)	Removes a socket handle from the list of polled sockets.

Legend

	Method
---	--------

2.11.1.5.1 - Constructors

2.11.1.5.1.1 - CPollSocket::CPollSocket Constructor

Default Constructor.

C++

```
CPollSocket();
```

Description

Constructor.

2.11.1.5.2 - Destructors

2.11.1.5.2.1 - CPollSocket::~CPollSocket Destructor

Destructor.

C++

```
virtual ~CPollSocket();
```

Description

Destructor.

2.11.1.5.3 - Methods

2.11.1.5.3.1 - CPollSocket::DisableEventsDetection Method

Disables the detection of specific events for a socket.

C++

```
mxt_result DisableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	Specifies the events to be disabled.

Returns

resS_OK: Success.

resFE_INVALID_ARGUMENT: hSocket is either invalid or not found in the socket pool or uEvents has an invalid value.

Description

Disables the detection of specific events for a socket. Updates the events to be detected. Only events that are true will be disabled. Events that are false are not enabled.

For example, if we have a new socket hSocket DisableEventDetection(hSocket, 0x06); DisableEventDetection(hSocket, 0x01); this will result in all events being disabled.

See Also

EnableEventsDetection (see page 542)

2.11.1.5.3.2 - CPollSocket::EnableEventsDetection Method

Enables the detection of specific events for a socket.

C++

```
mxt_result EnableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	Specifies the events to be enabled.

Returns

resS_OK: Success.

resFE_INVALID_ARGUMENT: hSocket is either invalid or not found in the socket pool or uEvents has an invalid value.

Description

Enables the detection of specific events for a socket. Updates the events to be detected. Only events that are true will be enabled. Events that are false are not disabled.

For example, if we have a new socket `hSocket` `EnableEventDetection(hSocket, 0x06);` `EnableEventDetection(hSocket, 0x01);` this will result in all events being enabled.

See Also

[DisableEventsDetection](#) (see page 542)

2.11.1.5.3.3 - CPollSocket::GetEventsDetectionState Method

Gets the events that are currently being detected.

C++

```
mxt_result GetEventsDetectionState(IN mxt_hSocket hSocket, OUT unsigned int* puEvents);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
OUT unsigned int* puEvents	On exit, will contain the current state of the event detection.

Returns

`resS_OK`: Success.

`resFE_INVALID_ARGUMENT`: `puEvents` is `NULL` or `hSocket` is invalid or not in the socket pool.

Description

Gets the events that are currently being detected.

See Also

[EnableEventsDetection](#) (see page 542), [DisableEventsDetection](#) (see page 542)

2.11.1.5.3.4 - CPollSocket::GetRegisteredSocketCount Method

Returns the number of sockets currently registered.

C++

```
unsigned int GetRegisteredSocketCount();
```

Returns

The number of registered sockets

Description

Gets the number of registered sockets in the poll.

2.11.1.5.3.5 - CPollSocket::Poll Method

This method is time critical and must be called periodically to detect events on registered sockets.

C++

```
mxt_result Poll(IN uint64_t uTimeoutMs = 0);
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs = 0	The maximum time the Poll method is allowed to block.

Returns

Description of the returned value.

Description

The `Poll()` method polls each registered socket to detect new events. If new events are detected, the associated manager is notified.

See Also

[IPollManager::EvPolledSocketMgrEventDetected](#)

2.11.1.5.3.6 - CPollSocket::RegisterSocket Method

Adds a socket handle / Polled socket manager association in the list of polled sockets.

C++

```
mxt_result RegisterSocket(IN mxt_hSocket hSocket, IN IPolledSocketMgr* pPolledSocketMgr, IN mxt_opaque opq = MX_INT32_TO_OPQ(0));
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be registered.
IN IPolledSocketMgr* pPolledSocketMgr	The manager that will be called when an event is detected on the socket.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value that will be provided to the manager.

Returns

resS_OK: Success.

resFE_INVALID_ARGUMENT: hSocket is either invalid, pPolledSocketMgr is NULL or already in the socket pool.

Description

Registers a socket handle for additional event detection.

See Also

UnregisterSocket (see page 544)

2.11.1.5.3.7 - CPollSocket::UnregisterSocket Method

Removes a socket handle from the list of polled sockets.

C++

```
mxt_result UnregisterSocket(IN mxt_hSocket hSocket, OUT mxt_opaque* popq = NULL);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be unregistered.
OUT mxt_opaque* popq = NULL	On exit, will contain the opaque value associated with the socket. May be NULL if the value should just be discarded.

Returns

resS_OK: Success.

resFE_INVALID_ARGUMENT: hSocket is either invalid or cannot be found in the socket pool.

Description

Unregister a socket.

See Also

RegisterSocket (see page 544)

2.11.1.6 - CSocket Class

Base socket class.

Class Hierarchy



C++

```
class CSocket;
```

Description

This is the base socket class. It implements sockets options and Windows Qos mechanism.

Location

Network/CSocket.h

2.11.1.7 - CSocketAddr Class

Class providing a common interface for storing socket addresses.

Class Hierarchy

CSocketAddr

C++

```
class CSocketAddr;
```

Description

The CSocketAddr class is used to provide a common interface to store socket addresses independently from the operating system and socket family used. This way all code used for sockets is the same for any architecture.

Constructors

Constructor	Description
CSocketAddr (see page 546)	Constructor. CSocketAddr

Legend

	Method
---	--------

Destructors

Destructor	Description
~CSocketAddr (see page 547)	Destructor. ~CSocketAddr

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 557)	Different than operator.
= (see page 557)	Assignment operator.
== (see page 558)	Comparison operator.
const sockaddr* (see page 558)	Cast operator.
sockaddr (see page 558)	Cast operator.
sockaddr* (see page 558)	Cast operator.
TInetAddr* (see page 559)	Cast operator.

Legend

	Method
---	--------

Methods

Method	Description
GetAddress (see page 548)	Gets the address. GetAddress
GetFamily (see page 549)	Gets the family.
GetPort (see page 549)	Gets the port.
GetScopeId (see page 550)	Gets the scope ID.
GetSize (see page 550)	Gets the size.
Inet6AnyAddress (see page 550)	Returns a INET6 CSocketAddr of type any.
InetAnyAddress (see page 550)	Returns a INET CSocketAddr of type any.
InetBroadcastAddress (see page 551)	Returns a INET CSocketAddr of type broadcast.
IsEqualAddress (see page 551)	Verifies that the addresses are equal.
IsEqualFamily (see page 551)	Verifies that the families are equal.

• IsEqualPort (see page 551)	Verifies that the port addresses are equal.
• IsEqualScope (see page 552)	Verifies that the addresses are of the same scope.
• IsEqualScopeld (see page 552)	Verifies that the scope ID are equal.
• IsInet6AddressGlobal (see page 552)	Verifies that the address is an IPv6 global address.
• IsInet6AddressLinkLocal (see page 552)	Verifies that the address is an IPv6 LinkLocal address.
• IsInet6AddressLoopback (see page 553)	Verifies that the address is the IPv6 loopback address.
• IsInet6AddressMulticast (see page 553)	Verifies that the address is an IPv6 multicast address.
• IsInet6AddressSiteLocal (see page 553)	Verifies that the address is an IPv6 SiteLocal address.
• IsInet6AddressUniqueLocal (see page 553)	Verifies that the address is an IPv6 unique local address (RFC4193).
• IsInet6AddressUnspecified (see page 553)	Verifies that the address is the IPv6 unspecified address.
• IsInet6AddressV4Mapped (see page 554)	Verifies that the address is a V4-mapped IPv6 address.
• IsInetAddressLoopback (see page 554)	Verifies that the port is the IPv4 loopback address.
• IsInetAddressMulticast (see page 554)	Verifies if the address is an IPv4 multicast address.
• IsValid (see page 554)	Verifies that the address and port in the object are valid.
• IsValidAddress (see page 554)	Verifies that the address is valid.
• IsValidFamily (see page 555)	Verifies that the address family is valid.
• IsValidPort (see page 555)	Verifies that the port is valid.
• Reset (see page 555)	Resets the object's data.
• SetAddress (see page 555)	Sets the address and port.
• SetPort (see page 557)	Sets the port.
• SetScopeld (see page 557)	Sets the scope ID.

Legend

•	Method
---	--------

2.11.1.7.1 - Constructors

2.11.1.7.1.1 - CSocketAddr

2.11.1.7.1.1.1 - CSocketAddr::CSocketAddr Constructor

Constructor.

CSocketAddr

C++

```
CSocketAddr();
```

Description

Constructor

2.11.1.7.1.1.2 - CSocketAddr::CSocketAddr Constructor

Constructor. Builds address using an address type and port number.

C++

```
CSocketAddr(IN EStandardAddress eType, IN uint16_t uPort = 0);
```

Parameters

Parameters	Description
IN EStandardAddress eType	Type of address.
IN uint16_t uPort = 0	The Port number.

Description

Constructor. Creates the socket address using the family, address type and port number

2.11.1.7.1.1.3 - CSocketAddr::CSocketAddr Constructor

Copy constructor.

C++

```
CSocketAddr(IN const CSocketAddr& rFrom);
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the address to copy.

Description

Copy constructor.

2.11.1.7.1.1.4 - CSocketAddr::CSocketAddr Constructor

Constructor. Builds address using a char array and port number.

C++

```
CSocketAddr(IN const char* pszAddress, IN uint16_t uPort = 0, IN EMxBase eBase = eBASE_AUTO, IN bool bForceFourParts = false);
```

Parameters

Parameters	Description
IN const char* pszAddress	Pointer to a string containing the dotted address. Must not be NULL, else it will assert.
IN uint16_t uPort = 0	The Port number.
IN EMxBase eBase = eBASE_AUTO	Base representation of the dotted address.
IN bool bForceFourParts = false	True if pszAddress must contain four parts (e.g. 1.2.3.4). False if pszAddress can be less than four parts (e.g. 100.256).

Description

Constructor. Creates the socket address from an address in dot form and a port number. If bForceFourParts is enabled, pszAddress must contain four integer parts separated by dots.

2.11.1.7.1.1.5 - CSocketAddr::CSocketAddr Constructor

Constructor. Builds address using a byte array, a family and a port number.

C++

```
CSocketAddr(IN const uint8_t* puAddress, IN unsigned int uAddressSize, IN EAddressFamily eFamily, IN uint16_t uPort = 0);
```

Parameters

Parameters	Description
IN const uint8_t* puAddress	Pointer to a byte array containing the address. Must not be NULL, else it will assert.
IN unsigned int uAddressSize	Size of the address buffer. If it's eINET family, the size must be g_uSIZE_OF_IPV4_ADDRESS. If it's eINET6 family, the size must be g_uSIZE_OF_IPV6_ADDRESS, else it will assert.
IN EAddressFamily eFamily	The family of the socket address. Must be eINET or eINET6, else it will assert
IN uint16_t uPort = 0	The Port number.

Description

Constructor. Creates the socket address using the address raw data and the port number.

Warning

Since the address is in raw data form, it MUST be in network byte order.

2.11.1.7.2 - Destructors**2.11.1.7.2.1 - CSocketAddr::~CSocketAddr Destructor**

Destructor.

~CSocketAddr (see page 545)

C++

```
virtual ~CSocketAddr();
```

Description

Destructor.

2.11.1.7.3 - Methods**2.11.1.7.3.1 - GetAddress****2.11.1.7.3.1.1 - CSocketAddr::GetAddress Method**

Gets the address.

GetAddress

C++

```
CString GetAddress(IN bool bShowScope = true) const;
```

Parameters

Parameters	Description
IN bool bShowScope = true	When true, will append the scope ID at the end of the address, delimited by the '%' symbol.

Returns

[CString (See page 126)] The address in the dotted decimal format for IPv4 addresses, and in hexadecimal format for IPv6 addresses.

Description

Copies the address in a CString (See page 126)

2.11.1.7.3.1.2 - CSocketAddr::GetAddress Method

Gets the address.

C++

```
mxt_result GetAddress(IN const unsigned int uAddressCapacity, OUT uint8_t* puAddress, OUT unsigned int* puAddressSize) const;
```

Parameters

Parameters	Description
IN const unsigned int uAddressCapacity	Capacity of puAddress.
OUT uint8_t* puAddress	Pointer to a buffer to contain the address in raw data format.
OUT unsigned int* puAddressSize	Pointer to contain the length of the address.

Returns

- resS_OK: The address was set successfully.
- resFE_INVALID_ARGUMENT: puAddress and/or puAddressSize are NULL, or the capacity of puAddress is too small.

Description

Gets the address in raw data format.

Warning

The data returned is in network byte order.

2.11.1.7.3.1.3 - CSocketAddr::GetAddress Method

Gets the address.

C++

```
mxt_result GetAddress(IN unsigned int uAddressSize, OUT char* pszAddress, IN bool bShowScope = true) const;
```

Parameters

Parameters	Description
IN unsigned int uAddressSize	Length of the provided buffer.
OUT char* pszAddress	Pointer to contain the dotted address.
IN bool bShowScope = true	When true, will append the scope ID at the end of the address, delimited by the '%' symbol.

Returns

- resS_OK: The address was set successfully.
- resFE_INVALID_ARGUMENT: the size of the given address is invalid.

Description

Gets the string representation of the address from the socket address. If resFE_INVALID_ARGUMENT is returned, the buffer content is invalid.

2.11.1.7.3.1.4 - CSocketAddr::GetAddress Method

Gets the address.

GetAddress()

C++

```
mxt_result GetAddress(OUT CString& rstrAddress, IN bool bShowScope = true) const;
```

Parameters

Parameters	Description
OUT CString& rstrAddress	Reference to the CString (see page 126) object where the address will be copied.
IN bool bShowScope = true	When true, will append the scope ID at the end of the address, delimited by the '%' symbol.

Returns

- resS_OK: The address was set successfully.
- resFE_INVALID_ARGUMENT: the size of the given address is invalid.

Description

Copies the IP address in a CString (see page 126). If resFE_INVALID_ARGUMENT is returned, the buffer content is invalid.

2.11.1.7.3.2 - CSocketAddr::GetFamily Method

Gets the family.

C++

```
EAddressFamily GetFamily() const;
```

Returns

The family type of the socket address.

Description

Gets the family type of the socket address.

2.11.1.7.3.3 - CSocketAddr::GetPort Method

Gets the port.

C++

```
uint16_t GetPort() const;
```

Returns

The port associated to this socket address.

Description

Gets the port associate to this socket address.

2.11.1.7.3.4 - CSocketAddr::GetScopId Method

Gets the scope ID.

C++

```
uint32_t GetScopeId() const;
```

Returns

- The Scope ID field of the sockaddr_in6 structure when the CSocketAddr (see page 545) for which this method was called an IPv6 address.
- 0 when the CSocketAddr (see page 545) for which this method was called is not an IPv6 address.

Description

Gets the scope ID.

2.11.1.7.3.5 - CSocketAddr::GetSize Method

Gets the size.

C++

```
unsigned int GetSize() const;
```

Returns

Size of the socket address structure.

Description

Gets the size of the socket address structure. Note that if MXD_IPV6_ENABLE_SUPPORT (see page 294) is not defined, GetSize will always return the size of an IPv4 address even if it contains an IPv6 address.

2.11.1.7.3.6 - CSocketAddr::Inet6AnyAddress Method

Returns a INET6 CSocketAddr (see page 545) of type any.

C++

```
static const CSocketAddr& Inet6AnyAddress();
```

Returns

A socket address of the INET6 family and type any.

Description

Creates a socket address of the INET6 family of the any type.

2.11.1.7.3.7 - CSocketAddr::InetAnyAddress Method

Returns a INET CSocketAddr (see page 545) of type any.

C++

```
static const CSocketAddr& InetAnyAddress();
```

Returns

A socket address of the INET family and type any.

Description

Creates a socket address of the INET family of the any type.

2.11.1.7.3.8 - CSocketAddr::InetBroadcastAddress Method

Returns a INET CSocketAddr (See page 545) of type broadcast.

C++

```
static const CSocketAddr& InetBroadcastAddress();
```

Returns

A socket address of the INET family and type broadcast.

Description

Creates a socket address of the INET family of the broadcast type.

2.11.1.7.3.9 - CSocketAddr::IsEqualAddress Method

Verifies that the addresses are equal.

C++

```
bool IsEqualAddress(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the socket address to compare with.

Returns

True if the two addresses are equal, false otherwise.

Description

Verifies if the two addresses are equal.

2.11.1.7.3.10 - CSocketAddr::IsEqualFamily Method

Verifies that the families are equal.

C++

```
bool IsEqualFamily(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the socket address to compare with.

Returns

True if the two families are different, false otherwise.

Description

Verifies if the two socket address families are different.

2.11.1.7.3.11 - CSocketAddr::IsEqualPort Method

Verifies that the port addresses are equal.

C++

```
bool IsEqualPort(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the socket address to compare with.

Returns

True if the two ports are equal, false otherwise.

Description

Verifies if the two ports are equal.

2.11.1.7.3.12 - CSocketAddr::IsEqualScope Method

Verifies that the addresses are of the same scope.

C++

```
bool IsEqualScope(IN CSocketAddr rFrom) const;
```

Parameters

Parameters	Description
IN CSocketAddr rFrom	Reference to the socket address to compare with.

Returns

True if the two addresses are of the same scope(i.e. Link-local, global), false otherwise.

Description

Verifies if the two addresses are of the same scope.

2.11.1.7.3.13 - CSocketAddr::IsEqualScopeld Method

Verifies that the scope ID are equal.

C++

```
bool IsEqualScopeld(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the socket address to compare with.

Returns

True if the two scope ID are equal, false otherwise.

Description

Verifies if the two scope ID are equal.

2.11.1.7.3.14 - CSocketAddr::IsInet6AddressGlobal Method

Verifies that the address is an IPv6 global address.

C++

```
bool IsInet6AddressGlobal() const;
```

Returns

True if the address is of type IPv6 global , false otherwise.

Description

Verifies if the address is an IPv6 global address.

2.11.1.7.3.15 - CSocketAddr::IsInet6AddressLinkLocal Method

Verifies that the address is an IPv6 LinkLocal address.

C++

```
bool IsInet6AddressLinkLocal() const;
```

Returns

True if the address is of type IPv6 link-local, false otherwise.

Description

Verifies if the address is of type IPv6 link-local.

2.11.1.7.3.16 - CSocketAddr::IsInet6AddressLoopback Method

Verifies that the address is the IPv6 loopback address.

C++

```
bool IsInet6AddressLoopback() const;
```

Returns

True if the address is of type IPv6 loopback, false otherwise.

Description

Verifies if the address is of type IPv6 loopback.

2.11.1.7.3.17 - CSocketAddr::IsInet6AddressMulticast Method

Verifies that the address is an IPv6 multicast address.

C++

```
bool IsInet6AddressMulticast() const;
```

Returns

True if the address is of type IPv6 multicast, false otherwise.

Description

Verifies if the address is of type IPv6 multicast.

2.11.1.7.3.18 - CSocketAddr::IsInet6AddressSiteLocal Method

Verifies that the address is an IPv6 SiteLocal address.

C++

```
bool IsInet6AddressSiteLocal() const;
```

Returns

True if the address is of type IPv6 site-local, false otherwise.

Description

Verifies if the address is of type IPv6 site-local.

2.11.1.7.3.19 - CSocketAddr::IsInet6AddressUniqueLocal Method

Verifies that the address is an IPv6 unique local address (RFC4193).

C++

```
bool IsInet6AddressUniqueLocal() const;
```

Returns

True if the address is of type IPv6 unique local , false otherwise.

Description

Verifies if the address is an IPv6 unique local address, as defined in RFC4193. IPv6 unique local addresses have the prefix FC00::/7.

2.11.1.7.3.20 - CSocketAddr::IsInet6AddressUnspecified Method

Verifies that the address is the IPv6 unspecified address.

C++

```
bool IsInet6AddressUnspecified() const;
```

Returns

True if the address is of type IPv6 unspecified, false otherwise.

Description

Verifies if the address is of type IPv6 unspecified.

2.11.1.7.3.21 - CSocketAddr::IsInet6AddressV4Mapped Method

Verifies that the address is a V4-mapped IPv6 address.

C++

```
bool IsInet6AddressV4Mapped() const;
```

Returns

True if the address is of type V4-mapped IPv6 , false otherwise.

Description

Verifies if the address is of type V4-mapped IPv6.

2.11.1.7.3.22 - CSocketAddr::IsInetAddressLoopback Method

Verifies that the port is the IPV4 loopback address.

C++

```
bool IsInetAddressLoopback() const;
```

Returns

True if the address is of type IPv4 loopback, false otherwise.

Description

Verifies if the address is of type IPv4 loopback.

2.11.1.7.3.23 - CSocketAddr::IsInetAddressMulticast Method

Verifies if the address is an IPv4 multicast address.

C++

```
bool IsInetAddressMulticast() const;
```

2.11.1.7.3.24 - CSocketAddr::IsValid Method

Verifies that the address and port in the object are valid.

C++

```
bool IsValid() const;
```

Returns

True if the socket address,port and family are valid, false otherwise.

Description

Verifies if the IP address, port and family are valid.

2.11.1.7.3.25 - CSocketAddr::IsValidAddress Method

Verifies that the address is valid.

C++

```
bool IsValidAddress() const;
```

Returns

True if the address is valid, false otherwise.

Description

Verifies if the address is valid.

2.11.1.7.3.26 - CSocketAddr::IsValidFamily Method

Verifies that the address family is valid.

C++

```
bool IsValidFamily() const;
```

Returns

True if the address family valid, false otherwise.

Description

Verifies if the address family is valid.

2.11.1.7.3.27 - CSocketAddr::IsValidPort Method

Verifies that the port is valid.

C++

```
bool IsValidPort() const;
```

Returns

True if the IP port is valid, false otherwise.

Description

Verifies if the IP port is valid.

2.11.1.7.3.28 - CSocketAddr::Reset Method

Resets the object's data.

C++

```
void Reset();
```

Description

Resets the address data to default values.

2.11.1.7.3.29 - SetAddress**2.11.1.7.3.29.1 - CSocketAddr::SetAddress Method**

Sets the address and port.

C++

```
mxt_result SetAddress(IN EStandardAddress eType, IN uint16_t uPort = 0);
```

Parameters

Parameters	Description
IN EStandardAddress eType	Type of socket address.
IN uint16_t uPort = 0	Port number.

Returns

- resS_OK: The address was set successfully.

Description

Sets the address to be contained within this object.

2.11.1.7.3.29.2 - CSocketAddr::SetAddress Method

Sets a CSocketAddr (see page 545) in the current object.

C++

```
mxt_result SetAddress(IN const CSocketAddr& rAddress);
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddress	Reference to the socket address to set.

Description

Sets a socket address to this one.

2.11.1.7.3.29.3 - CSocketAddr::SetAddress Method

Sets the address and port.

C++

```
mxt_result SetAddress(IN const char* pszAddress, IN uint16_t uPort = 0, IN EMxBase eBase = eBASE_AUTO, IN bool bForceFourParts = false);
```

Parameters

Parameters	Description
IN const char* pszAddress	Pointer to the dotted address to set.
IN uint16_t uPort = 0	Port number.
IN EMxBase eBase = eBASE_AUTO	Base representation of the dotted address.
IN bool bForceFourParts = false	True if pszAddress must contain four parts (e.g. 1.2.3.4). False if pszAddress can be less than four parts (e.g. 100.256).

Returns

- resS_OK: The address was set successfully.
- resFE_INVALID_ARGUMENT: The address given was invalid.

Description

Sets the address to be contained within this object. If bForceFourParts is enabled, pszAddress must contain four integer parts separated by dots.

2.11.1.7.3.29.4 - CSocketAddr::SetAddress Method

Sets the address and port.

C++

```
mxt_result SetAddress(IN const uint8_t* puAddress, IN unsigned int uAddressSize, IN EAddressFamily eFamily, IN uint16_t uPort = 0);
```

Parameters

Parameters	Description
IN const uint8_t* puAddress	Pointer to the address in raw data.
IN unsigned int uAddressSize	Size of the buffer.
IN EAddressFamily eFamily	Family of the address.
IN uint16_t uPort = 0	Port number.

Returns

- resS_OK: The address was set successfully.
- resFE_INVALID_ARGUMENT: the size of the given address is invalid.

Description

Sets the address to be contained within this object.

Notes

The data pointed to by puAddress MUST be in network byte order. Also, it represent the network address in raw data.

2.11.1.7.3.30 - CSocketAddr::SetPort Method

Sets the port.

C++

```
void SetPort(IN uint16_t uPort);
```

Parameters

Parameters	Description
IN uint16_t uPort	Port number.

Description

Sets the port.

2.11.1.7.3.31 - CSocketAddr::SetScopeId Method

Sets the scope ID.

C++

```
mxt_result SetScopeId(IN uint32_t uScopeId);
```

Parameters

Parameters	Description
IN uint32_t uScopeId	Scope ID.

Returns

- resS_OK: The scope ID was set successfully.
- resFE_INVALID_STATE: The CSocketAddr (see page 545) for which this method was called is not an IPv6 address.

Description

Sets the scope ID field in the sockaddr_in6 structure.

2.11.1.7.4 - Operators**2.11.1.7.4.1 - CSocketAddr::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the socket address to compare with.

Returns

True if the two socket addresses are different, false otherwise.

Description

Verifies if the two socket addresses are different.

2.11.1.7.4.2 - CSocketAddr::= Operator

Assignment operator.

C++

```
CSocketAddr& operator =(IN const CSocketAddr& rFrom);
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the address to copy.

Returns

Reference to the assigned address.

Description

Assignment operator. Assigns the right hand address to the left hand one.

2.11.1.7.4.3 - CSocketAddr::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CSocketAddr& rFrom) const;
```

Parameters

Parameters	Description
IN const CSocketAddr& rFrom	Reference to the address to compare.

Returns

True if both addresses are equal, false otherwise.

Description

Verifies is both addresses are equal.

2.11.1.7.4.4 - CSocketAddr::const sockaddr* Operator

Cast operator.

C++

```
operator const sockaddr*() const;
```

Description

Casts the current object into a const sockaddr (see page 558) pointer.

2.11.1.7.4.5 - CSocketAddr::sockaddr Operator

Cast operator.

C++

```
operator sockaddr() const;
```

Description

Casts the current object into a const sockaddr.

2.11.1.7.4.6 - CSocketAddr::sockaddr* Operator

Cast operator.

C++

```
operator sockaddr*() const;
```

Description

Casts the current object into a sockaddr (see page 558) pointer.

2.11.1.7.4.7 - CSocketAddr::TInetAddr* Operator Symbian OS only

Cast operator.

C++

```
operator TInetAddr*() const;
```

2.11.1.8 - CTcpServerSocket Class

Implements a TCP socket in servermode.

Class Hierarchy



C++

```
class CTcpServerSocket : protected CSocket, public IServerSocket;
```

Description

The CTcpServerSocket class is an implementation of a TCP socket in server mode which supports the IAServerSocket interface.

Location

Network/CTcpServerSocket.h

Constructors

Constructor	Description
CTcpServerSocket (see page 560)	Default Constructor.

Legend

	Constructor
	Method

Methods

Method	Description
Accept (see page 560)	Accepts the connection request from a client socket.
Bind (see page 561)	Binds the socket to a local address.
Close (see page 561)	Closes a socket.
Create (see page 562)	Creates the socket.
GetAddressFamily (see page 562)	Retrieves the address family of this socket.
GetHandle (see page 562)	Retrieve the socket handle.
GetLocalAddress (see page 563)	Retrieves the local address to which the socket is bound.
GetProtocolFamily (see page 563)	Retrieves the protocol family of this socket.
GetSocketType (see page 563)	Retrieves information about the socket and its transport.
Listen (see page 564)	Lists on the socket for incoming connection attempts.
Release (see page 564)	Deletes this socket.
Set8021QUserPriority (see page 564)	Set 8021Q user priority option.
SetBlocking (see page 565)	Set socket operation to blocking/non-blocking
SetIpv6UnicastHops (see page 565)	Sets the IPv6 outgoing unicast hop limit.
SetKeepAlive (see page 565)	Enable/disable the keep-alive option.
SetReuseAddress (see page 565)	Enable/disable the possibility to reuse a local address.
SetTos (see page 566)	Set socket type of service (TOS byte) option
SetWindowsReceivingFlowspec (see page 566)	Sets the receiving flowspec in windows only.
SetWindowsSendingFlowspec (see page 566)	Sets the sending flowspec in windows only.

IServerSocket Class

IServerSocket Class	Description
Accept (see page 645)	Accepts an incoming connection attempt.
Bind (see page 645)	Binds the socket to a local address.
Listen (see page 646)	Lists on the socket for incoming connection attempts.

Legend

	Method
	virtual
	abstract

2.11.1.8.1 - Constructors

Default Constructor.

C++

```
CTcpServerSocket();
```

Description

Constructor.

2.11.1.8.2 - Methods**2.11.1.8.2.1 - Accept****2.11.1.8.2.1.1 - CTcpServerSocket::Accept Method**

Accepts the connection request from a client socket.

C++

```
mxt_result Accept(OUT GO CTcpSocket** ppTcpSocket);
```

Parameters

Parameters	Description
OUT GO CTcpSocket** ppTcpSocket	OUT (see page 39) parameter used to receive the pointer to a CTcpSocket (see page 567) representing the accepted connection. Note that the caller will receive the ownership of this pointer. This means that it will be the responsibility of the caller to delete this pointer by calling Release (see page 564) on it (unless the ownership of this pointer is given to some other entity).

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

Accepts an incoming connection attempt on a listening connection-oriented socket.

Please take note of the ownership note given in the parameter description.

See Also

Listen (see page 564)

2.11.1.8.2.1.2 - CTcpServerSocket::Accept Method

Accepts an incoming connection attempt.

C++

```
virtual mxt_result Accept(OUT GO IIoSocket** ppIoSocket);
```

Parameters

Parameters	Description
ppIoSocket	OUT parameter used to receive the pointer to a IIoSocket representing the accepted connection. Note that the caller will receive the ownership of this pointer. This means that it will be the responsibility of the caller to delete this pointer by calling Release on it (unless the ownership of this pointer is given to some other entity).

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

Accepts an incoming connection attempt on a listening connection-oriented socket.

Please take note of the ownership note given in the parameter description.

See Also

Listen

2.11.1.8.2.2 - CTcpServerSocket::Bind Method

Binds the socket to a local address.

C++

```
virtual mxt_result Bind(IN const CSocketAddr* pLocalAddress, OUT CSocketAddr* pEffectiveLocalAddress);
```

Parameters

Parameters	Description
pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.
pEffectiveLocalAddress	Upon return, contains the address to which the socket has effectively been bound. It may be NULL if the effective binding is not important to the caller.

Returns

- resS_OK: Socket has been bound.
- Failure codes: Socket has not been bound.

Description

Used to associate a local address to the socket.

2.11.1.8.2.3 - CTcpServerSocket::Close Method

Closes a socket.

C++

```
virtual mxt_result Close(IN ECloseBehavior eBehavior);
```

Parameters

Parameters	Description
eBehavior	How the socket is to be closed, as defined by the ECloseBehavior enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to close a socket. This puts the socket into the state it was just after its allocation. After this call, all system resources associated to the socket must have been released. Note that a socket can be reused after it has been closed. In such a case,

the socket options are all back to their default values. In other words, calling Close() on a socket has the effect of resetting the socket options to their default values after it is closed.

2.11.1.8.2.4 - CTcpServerSocket::Create Method

Creates the socket.

C++

```
mxt_result Create(IN CSocketAddr::EAddressFamily eAddressFamily);
```

Parameters

Parameters	Description
IN CSocketAddr::EAddressFamily eAddressFamily	The address family to use.

Returns

- resFE_INVALID_STATE: Already created.
- Errors returned by GetSocketErrorId (see page 656)().
- resFE_NOT_IMPLEMENTED: Protocol family not supported.
- resS_OK: Method processed successfully.

Description

This method is used to create a socket to communicate over the TCP protocol as a server using the protocol family corresponding to the address family parameter. This will only allocate all system resources required for the socket to communicate. Any other methods called before Create() will result in an error.

2.11.1.8.2.5 - CTcpServerSocket::GetAddressFamily Method

Retrieves the address family of this socket.

C++

```
virtual mxt_result GetAddressFamily(OUT CSocketAddr::EAddressFamily* peAddressFamily) const;
```

Parameters

Parameters	Description
peAddressFamily	OUT parameter that receives the address family, as defined in the CSocketAddr::EAddressFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address family associated to the socket.

2.11.1.8.2.6 - CTcpServerSocket::GetHandle Method

Retrieve the socket handle.

C++

```
mxt_hSocket GetHandle() const;
```

Returns

The socket's handle.

Description

This method is used to retrieve the socket handle.

2.11.1.8.2.7 - CTcpServerSocket::GetLocalAddress Method

Retrieves the local address to which the socket is bound.

C++

```
virtual mxt_result GetLocalAddress(OUT CSocketAddr* pLocalAddress) const;
```

Parameters

Parameters	Description
pLocalAddress	OUT parameter that will contain the address and port information.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address and port to which the socket has been bound. It will return an error until the socket is bound to an interface with the Bind function.

2.11.1.8.2.8 - CTcpServerSocket::GetProtocolFamily Method Deprecated since v2.1.4

Retrieves the protocol family of this socket.

C++

```
virtual mxt_result GetProtocolFamily(OUT EProtocolFamily* peProtocolFamily) const;
```

Parameters

Parameters	Description
peProtocolFamily	OUT parameter that receives the protocol family, as defined in the EProtocolFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the protocol family associated to the socket. This method is deprecated and GetAddressFamily should be used instead.

2.11.1.8.2.9 - CTcpServerSocket::GetSocketType Method

Retrieves information about the socket and its transport.

C++

```
virtual mxt_result GetSocketType(OUT ESocketType* peSocketType) const;
```

Parameters

Parameters	Description
peSocketType	OUT parameter that receives the socket type information, as defined in the ESocketType enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve information about this socket and its transport.

2.11.1.8.2.10 - CTcpServerSocket::Listen Method

Listens on the socket for incoming connection attempts.

C++

```
virtual mxt_result Listen(IN unsigned int uMaxPendingConnection = 5);
```

Parameters

Parameters	Description
uMaxPendingConnection	Maximum number of pending connections accepted.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

Sets the socket to listen for incoming connection attempts. Once a connection attempt is detected, Accept must be called in order to fully establish the connection.

See Also

Accept

2.11.1.8.2.11 - CTcpServerSocket::Release Method

Deletes this socket.

C++

```
virtual uint32_t Release();
```

Returns

Always zero.

Description

This method is used to delete a socket. This has become necessary in order to be able to delete the socket from one of its interfaces (instead of directly calling the class that implements these interfaces). This is called by the user of the socket instead of the delete operator.

Note that this can be called if the socket is not closed yet. In such a case, the socket will be closed automatically before the socket is deleted.

The return value is always zero. We eventually plan to implement reference counting on the sockets, and the reference count number will be returned instead of zero when calling Release.

2.11.1.8.2.12 - CTcpServerSocket::Set8021QUserPriority Method

Set 8021Q user priority option.

C++

```
mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority);
```

Parameters

Parameters	Description
bEnable	Determine whether the option is enable or not.
uUserPriority	The value to use in the 8021Q user priority field.

Returns

- resS_OK: The value has been correctly set.

- results returned by SetSockOpt8021QUserPriority or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.8.2.13 - **CTcpServerSocket::SetBlocking** Method

Set socket operation to blocking/non-blocking

C++

```
mxt_result SetBlocking(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the socket operation mode is blocking (true) or not (false).

Returns

- Results returned by SetSockOptBlocking.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.8.2.14 - **CTcpServerSocket::SetIpv6UnicastHops** Method

Sets the IPv6 outgoing unicast hop limit.

C++

```
mxt_result SetIpv6UnicastHops(IN int nHops);
```

Parameters

Parameters	Description
nHops	Number of hops.

Returns

- Results returned by SetSockOptIpv6UnicastHops.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.8.2.15 - **CTcpServerSocket::SetKeepAlive** Method

Enable/disable the keep-alive option.

C++

```
mxt_result SetKeepAlive(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Enable/disable the keep-alive option.

Returns

- Results returned by SetSockOptKeepAliveEnable.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

See Also

[SetSockOptKeepAliveEnable](#)

2.11.1.8.2.16 - **CTcpServerSocket::SetReuseAddress** Method

Enable/disable the possibility to reuse a local address.

C++

```
mxt_result SetReuseAddress(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the option is enable (true) or not (false).

Returns

- Results returned by SetSockOptReuseAddress.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.8.2.17 - CTcpServerSocket::SetTos Method

Set socket type of service (TOS byte) option

C++

```
mxt_result SetTos(IN uint8_t uTos);
```

Parameters

Parameters	Description
uTos	The value of the TOS byte field for this socket.

Returns

- resS_OK
- Results returned by SetSockOptTos or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.8.2.18 - CTcpServerSocket::SetWindowsReceivingFlowspec Method

Sets the receiving flowspec in windows only.

C++

```
mxt_result SetWindowsReceivingFlowspec(IN FLOWSPEC* pReceivingFlowspec);
```

Parameters

Parameters	Description
pReceivingFlowspec	Pointer to the receiving FLOWSPEC to set.

Returns

- resS_OK

Description

Sets the receiving flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.8.2.19 - CTcpServerSocket::SetWindowsSendingFlowspec Method

Sets the sending flowspec in windows only.

C++

```
mxt_result SetWindowsSendingFlowspec(IN FLOWSPEC* pSendingFlowspec);
```

Parameters

Parameters	Description
pSendingFlowspec	Pointer to the sending FLOWSPEC to set.

Returns

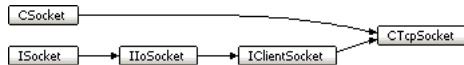
- resS_OK

Description

Sets the sending flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.9 - CTcpSocket Class

Implementation of a client TCP socket.

Class Hierarchy**C++**

```
class CTcpSocket : protected CSocket, public IClientSocket;
```

Description

The CTcpSocket class is an implementation of a TCP socket in client mode which supports the IClientSocket (see page 635) interface.

Location

Network/CTcpSocket.h

Constructors

Constructor	Description
CTcpSocket (see page 568)	Default Constructor.

Legend**Methods**

Method	Description
Bind (see page 568)	Binds the socket to a local address.
Close (see page 568)	Closes a socket.
Connect (see page 569)	Connects the socket to a remote address.
Create (see page 569)	Creates the socket.
GetAddressFamily (see page 570)	Retrieves the address family of this socket.
GetHandle (see page 570)	Retrieve the socket handle.
GetLocalAddress (see page 570)	Retrieves the local address to which the socket is bound.
GetPeerAddress (see page 571)	Retrieves the remote address to which the socket is connected.
GetProtocolFamily (see page 571)	Retrieves the protocol family of this socket.
GetSocketType (see page 571)	Retrieves information about the socket and its transport.
Recv (see page 572)	Receives data from a connected socket.
RecvFrom (see page 573)	Receives data from a non-connected socket.
Release (see page 574)	Deletes this socket.
Send (see page 574)	Sends data on a connected socket.
SendTo (see page 575)	Sends data on a non-connected socket.
Set8021QUserPriority (see page 576)	Set 8021Q user priority option.
SetBlocking (see page 576)	Set socket operation to blocking/non-blocking
SetIpv6UnicastHops (see page 577)	Sets the IPv6 outgoing unicast hop limit.
SetKeepAlive (see page 577)	Enable/disable the keep-alive option.
SetNagle (see page 577)	Enable/disable the Nagle algorithm.
SetReceiveBufferSize (see page 578)	Set socket receive buffer size option
SetReuseAddress (see page 578)	Enable/disable the possibility to reuse a local address.
SetTos (see page 578)	Set socket type of service (TOS byte) option
SetTransmitBufferSize (see page 579)	Set socket transmit buffer size option
SetWindowsReceivingFlowspec (see page 579)	Sets the receiving flowspec in windows only.
SetWindowsSendingFlowspec (see page 579)	Sets the sending flowspec in windows only.

IClientSocket Class

IClientSocket Class	Description
Bind (See page 636)	Binds the socket to a local address.
Connect (See page 637)	Connects the socket to a remote address.

Legend

 M	Method
 V	virtual
 A	abstract

2.11.1.9.1 - Constructors**2.11.1.9.1.1 - CTcpSocket::CTcpSocket Constructor**

Default Constructor.

C++

```
CTcpSocket();
```

Description

Constructor.

2.11.1.9.2 - Methods**2.11.1.9.2.1 - CTcpSocket::Bind Method**

Binds the socket to a local address.

C++

```
virtual mxt_result Bind(IN const CSocketAddr* pLocalAddress, OUT CSocketAddr* pEffectiveLocalAddress);
```

Parameters

Parameters	Description
pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.
pEffectiveLocalAddress	Upon return, contains the address to which the socket has effectively been bound. It may be NULL if the effective binding is not important to the caller.

Returns

- resS_OK: Socket has been bound.
- Failure codes: Socket has not been bound.

Description

Used to associate a local address to the socket.

2.11.1.9.2.2 - CTcpSocket::Close Method

Closes a socket.

C++

```
virtual mxt_result Close(IN ECloseBehavior eBehavior);
```

Parameters

Parameters	Description
eBehavior	How the socket is to be closed, as defined by the ECloseBehavior enumeration.

Returns

- resFE_INVALID_STATE

- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to close a socket. This puts the socket into the state it was just after its allocation. After this call, all system resources associated to the socket must have been released. Note that a socket can be reused after it has been closed. In such a case, the socket options are all back to their default values. In other words, calling Close() on a socket has the effect of resetting the socket options to their default values after it is closed.

2.11.1.9.2.3 - CTcpSocket::Connect Method

Connects the socket to a remote address.

C++

```
virtual mxt_result Connect(IN const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pPeerAddress	Remote address where the socket is to be connected.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to connect the socket to a remote address. Once a socket is connected, Send and Recv can be used to exchange data with the remote peer.

2.11.1.9.2.4 - Create

2.11.1.9.2.4.1 - CTcpSocket::Create Method

Creates the socket.

C++

```
mxt_result Create(IN CSocketAddr::EAddressFamily eAddressFamily);
```

Parameters

Parameters	Description
IN CSocketAddr::EAddressFamily eAddressFamily	The address family to use.

Returns

- resS_OK
- resFE_INVALID_STATE: Already created.
- Errors returned by GetSocketErrorId (see page 656)().
- resFE_NOT_IMPLEMENTED: Protocol family not supported.

Description

This method is used to create a socket to communicate over the TCP protocol as a client using the protocol family corresponding to the specified address family. This will only allocate all system resources required for the socket to communicate. Any other methods called before Create will result in an error.

2.11.1.9.2.4.2 - CTcpSocket::Create Method

Also creates the socket but this method is restricted to CTcpServerSocket::Accept (see page 560)() method.

C++

```
mxt_result Create(IN EProtocolFamily eProtocolFamily, IN mxt_hSocket hSocket, IN CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
IN EProtocolFamily eProtocolFamily	The protocol family to be used by the socket as defined by the EProtocolFamily (see page 651) enumeration.
IN mxt_hSocket hSocket	The socket handle of the new socket
IN CSocketAddr* pPeerAddress	The peer address to communicate with.

Returns

- resS_OK
- resFE_INVALID_STATE: Already created.
- resFE_NOT_IMPLEMENTED: Protocol family not supported.
- resFE_INVALID_ARGUMENT: hSocket is invalid.

Description

This method is used to create a socket to communicate over the TCP protocol as a client using the protocol family specified by eProtocolFamily enumeration. This will only allocate all system resources required for the socket to communicate. Any other methods called before Create will result in an error. This method can only be used by the CTcpServerSocket::Accept (see page 560) method. Normally, pPeerAddress will contain the address of the server accepting a client request.

2.11.1.9.2.5 - CTcpSocket::GetAddressFamily Method

Retrieves the address family of this socket.

C++

```
virtual mxt_result GetAddressFamily(OUT CSocketAddr::EAddressFamily* peAddressFamily) const;
```

Parameters

Parameters	Description
peAddressFamily	OUT parameter that receives the address family, as defined in the CSocketAddr::EAddressFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address family associated to the socket.

2.11.1.9.2.6 - CTcpSocket::GetHandle Method

Retrieve the socket handle.

C++

```
mxt_hSocket GetHandle() const;
```

Returns

The socket's handle.

Description

This method is used to retrieve the socket handle.

2.11.1.9.2.7 - CTcpSocket::GetLocalAddress Method

Retrieves the local address to which the socket is bound.

C++

```
virtual mxt_result GetLocalAddress(OUT CSocketAddr* pLocalAddress) const;
```

Parameters

Parameters	Description
pLocalAddress	OUT parameter that will contain the address and port information.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address and port to which the socket has been bound. It will return an error until the socket is bound to an interface with the Bind function.

2.11.1.9.2.8 - CTcpSocket::GetPeerAddress Method

Retrieves the remote address to which the socket is connected.

C++

```
virtual mxt_result GetPeerAddress(OUT CSocketAddr* pPeerAddress) const;
```

Parameters

Parameters	Description
pPeerAddress	OUT parameter used to receive the address.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address to which the socket is currently connected. It can return an error for unconnected sockets.

2.11.1.9.2.9 - CTcpSocket::GetProtocolFamily Method Deprecated since v2.1.4

Retrieves the protocol family of this socket.

C++

```
virtual mxt_result GetProtocolFamily(OUT EProtocolFamily* peProtocolFamily) const;
```

Parameters

Parameters	Description
peProtocolFamily	OUT parameter that receives the protocol family, as defined in the EProtocolFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the protocol family associated to the socket. This method is deprecated and GetAddressFamily should be used instead.

2.11.1.9.2.10 - CTcpSocket::GetSocketType Method

Retrieves information about the socket and its transport.

C++

```
virtual mxt_result GetSocketType(OUT ESocketType* peSocketType) const;
```

Parameters

Parameters	Description
peSocketType	OUT parameter that receives the socket type information, as defined in the ESocketType enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve information about this socket and its transport.

2.11.1.9.2.11 - Recv**2.11.1.9.2.11.1 - CTcpSocket::Recv Method**

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT CBlob* pData);
```

Parameters

Parameters	Description
pData	Blob object that will be filled with the data received on the socket.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from a connected socket.

The maximum number of bytes that can be received is defined by the blob's capacity. The user of the socket might want to resize the blob to a specific size before calling Recv, as its capacity will not be changed by this call.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::Recv
- IloSocket::RecvFrom

2.11.1.9.2.11.2 - CTcpSocket::Recv Method

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize);
```

Parameters

Parameters	Description
puData	Pointer to a buffer where the received data is to be stored.
uCapacity	The maximum size available in the buffer pointed to by puData.
puSize	OUT parameter that contains the number of bytes actually written in the buffer.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from a connected socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::Recv
- IloSocket::RecvFrom

2.11.1.9.2.12 - RecvFrom**2.11.1.9.2.12.1 - CTcpSocket::RecvFrom Method**

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT CBlob* pData, OUT CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pData	Blob object that will be filled with the data received on the socket.
pPeerAddress	Address from which the data was received.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::RecvFrom
- IloSocket::Recv

2.11.1.9.2.12.2 - CTcpSocket::RecvFrom Method

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize, OUT CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
puData	Pointer to a buffer where the received data is to be stored.
uCapacity	The maximum size available in the buffer pointed to by puData.
puSize	OUT parameter that contains the number of bytes actually written in the buffer.
pPeerAddress	Address from which the data was received.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::RecvFrom
- IloSocket::Recv

2.11.1.9.2.13 - CTcpSocket::Release Method

Deletes this socket.

C++

```
virtual uint32_t Release();
```

Returns

Always zero.

Description

This method is used to delete a socket. This has become necessary in order to be able to delete the socket from one of its interfaces (instead of directly calling the class that implements these interfaces). This is called by the user of the socket instead of the delete operator.

Note that this can be called if the socket is not closed yet. In such a case, the socket will be closed automatically before the socket is deleted.

The return value is always zero. We eventually plan to implement reference counting on the sockets, and the reference count number will be returned instead of zero when calling Release.

2.11.1.9.2.14 - Send**2.11.1.9.2.14.1 - CTcpSocket::Send Method**

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const CBlob* pData, OUT unsigned int* puSizeSent);
```

Parameters

Parameters	Description
pData	Pointer to a blob object that contains the data to be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IIoSocket::Send
- IIoSocket::SendTo

2.11.1.9.2.14.2 - CTcpSocket::Send Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent);
```

Parameters

Parameters	Description
puData	Pointer to a buffer containing the data to send.
uSize	The number of bytes in the buffer that must be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IIoSocket::Send
- IIoSocket::SendTo

2.11.1.9.2.15 - SendTo

2.11.1.9.2.15.1 - CTcpSocket::SendTo Method

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const CBlob* pData, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pData	Pointer to a blob containing the data to be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
pPeerAddress	Address to which the data must be sent.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::SendTo
- IloSocket::Send

2.11.1.9.2.15.2 - CTcpSocket::SendTo Method

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent, IN
                           const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
puData	Pointer to a buffer containing the data to send.
uSize	The number of bytes in the buffer that must be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
pPeerAddress	Address to which the data must be sent.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::SendTo
- IloSocket::Send

2.11.1.9.2.16 - CTcpSocket::Set8021QUserPriority Method

Set 8021Q user priority option.

C++

```
mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority);
```

Parameters

Parameters	Description
bEnable	Determine whether the option is enable or not.
uUserPriority	The value to use in the 8021Q user priority field.

Returns

- resS_OK: The value has been correctly set.
- results returned by SetSockOpt8021QUserPriority or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.17 - CTcpSocket::SetBlocking Method

Set socket operation to blocking/non-blocking

C++

```
mxt_result SetBlocking(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the socket operation mode is blocking (true) or not (false).

Returns

- Results returned by SetSockOptBlocking.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.18 - CTcpSocket::SetIpv6UnicastHops Method

Sets the IPv6 outgoing unicast hop limit.

C++

```
mxt_result SetIpv6UnicastHops(IN int nHops);
```

Parameters

Parameters	Description
nHops	Number of hops.

Returns

- Results returned by SetSockOptIpv6UnicastHops.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.19 - CTcpSocket::SetKeepAlive Method

Enable/disable the keep-alive option.

C++

```
mxt_result SetKeepAlive(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Enable/disable the keep-alive option.

Returns

- Results returned by SetSockOptKeepAliveEnable.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

See Also

[SetSockOptKeepAliveEnable](#)

2.11.1.9.2.20 - CTcpSocket::SetNagle Method

Enable/disable the Nagle algorithm.

C++

```
mxt_result SetNagle(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Enable if true / Disable if false.

Returns

- Results returned by SetSockOptNagle.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.21 - CTcpSocket::SetReceiveBufferSize Method

Set socket receive buffer size option

C++

```
mxt_result SetReceiveBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the receive buffer.

Returns

- Results returned by SetSockOptReceiveBufferSize.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option. Usually, the default value of this option is adequate. Do not modify it unless you know what you do.

2.11.1.9.2.22 - CTcpSocket::SetReuseAddress Method

Enable/disable the possibility to reuse a local address.

C++

```
mxt_result SetReuseAddress(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the option is enable (true) or not (false).

Returns

- Results returned by SetSockOptReuseAddress.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.23 - CTcpSocket::SetTos Method

Set socket type of service (TOS byte) option

C++

```
mxt_result SetTos(IN uint8_t uTos);
```

Parameters

Parameters	Description
uTos	The value of the TOS byte field for this socket.

Returns

- resS_OK
- Results returned by SetSockOptTos or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.9.2.24 - CTcpSocket::SetTransmitBufferSize Method

Set socket transmit buffer size option

C++

```
mxt_result SetTransmitBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the transmit buffer.

Returns

- Results returned by SetSockOptTransmitBufferSize.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option. Usually, the default value of this option is adequate. Do not modify it unless you know what you do.

2.11.1.9.2.25 - CTcpSocket::SetWindowsReceivingFlowspec Method

Sets the receiving flowspec in windows only.

C++

```
mxt_result SetWindowsReceivingFlowspec(IN FLOWSPEC* pReceivingFlowspec);
```

Parameters

Parameters	Description
pReceivingFlowspec	Pointer to the receiving FLOWSPEC to set.

Returns

- resS_OK

Description

Sets the receiving flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.9.2.26 - CTcpSocket::SetWindowsSendingFlowspec Method

Sets the sending flowspec in windows only.

C++

```
mxt_result SetWindowsSendingFlowspec(IN FLOWSPEC* pSendingFlowspec);
```

Parameters

Parameters	Description
pSendingFlowspec	Pointer to the sending FLOWSPEC to set.

Returns

- resS_OK

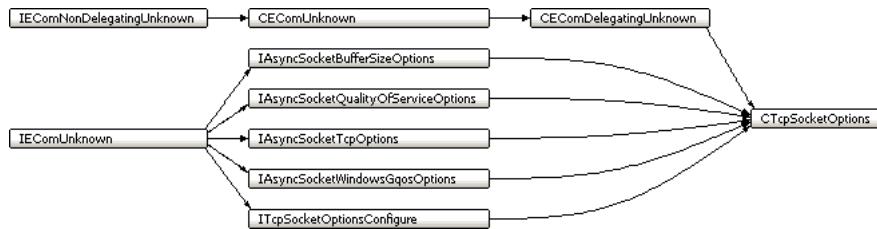
Description

Sets the sending flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.10 - CTcpSocketOptions Class

An ECOM (see page 412) object that can be used to configure TCP socket options.

Class Hierarchy



C++

```

class CTcpSocketOptions : public CEComDelegatingUnknown, public IAsyncSocketBufferSizeOptions, public
IAsyncSocketQualityOfServiceOptions, public IAsyncSocketTcpOptions, public IAsyncSocketWindowsGqosOptions,
public ITcpSocketOptionsConfigure;
  
```

Description

An ECOM (see page 412) object that can be used to configure TCP socket options. Until a TCP socket is specified via ITcpSocketOptionsConfigure (see page 650), options set via other interfaces are cached.

Location

Network/CTcpSocketOptions.h

Constructors

CEComDelegatingUnknown Class

CEComDelegatingUnknown Class	Description
• CEComDelegatingUnknown (see page 414)	Constructor

CEComUnknown Class

CEComUnknown Class	Description
• CEComUnknown (see page 415)	Constructor.

Legend

•	Method
---	--------

Destructors

CEComDelegatingUnknown Class

CEComDelegatingUnknown Class	Description
• ~CEComDelegatingUnknown (see page 414)	Destructor

CEComUnknown Class

CEComUnknown Class	Description
• ~CEComUnknown (see page 416)	Destructor.

Legend

•	Method
▼	virtual

Methods

Method	Description
• V ApplyOptions (see page 581)	Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.
• V CreateInstance (see page 581)	This is the ECOM (see page 412) Creation handler.
• V NonDelegatingQueryIf (see page 582)	The following statement is essential, it provide the default and unique implementation of the IEComUnknown (see page 416) interface that every other interfaces inherit from.
• V Set8021QUserPriority (see page 582)	Sets a socket level option which enables/disables utilization of the 802.1Q user priority.
• V SetConnectTimeoutMs (see page 582)	Sets the connect timeout in ms.
• V SetKeepAlive (see page 582)	Enables/disables the TCP keep-alive option.
• V SetNagle (see page 583)	Enables/disables the TCP Nagle algorithm option.

•   SetReceiveBufferSize (see page 583)	Sets a socket level option which determines the maximum socket receive buffer size.
•   SetTos (see page 583)	Sets a socket level option which sets the TOS byte field in the IP protocol header.
•   SetTransmitBufferSize (see page 584)	Sets a socket level option which determines the maximum socket transmit buffer size.
•   SetWindowsReceivingFlowspec (see page 584)	Sets the receiving flowspec (Windows only).
•   SetWindowsSendingFlowspec (see page 584)	Sets the sending flowspec (Windows only).

ITcpSocketOptionsConfigure Class

ITcpSocketOptionsConfigure Class	Description
•   ApplyOptions (see page 650)	Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

IAsyncSocketWindowsGqosOptions Class

IAsyncSocketWindowsGqosOptions Class	Description
•   SetWindowsReceivingFlowspec (see page 630)	Sets the receiving flowspec (Windows only).
•   SetWindowsSendingFlowspec (see page 630)	Sets the sending flowspec (Windows only).

IAsyncSocketTcpOptions Class

IAsyncSocketTcpOptions Class	Description
•   SetConnectTimeoutMs (see page 628)	Sets the connect timeout in ms.
•   SetKeepAlive (see page 628)	Enables/disables the TCP keep-alive option.
•   SetNagle (see page 628)	Enables/disables the TCP Nagle algorithm option.

IAsyncSocketQualityOfServiceOptions Class

IAsyncSocketQualityOfServiceOptions Class	Description
•   Set8021QUserPriority (see page 626)	Sets a socket level option which enables/disables utilization of the 802.1Q user priority.
•   SetTos (see page 627)	Sets a socket level option which sets the TOS (see page 39) byte field in the IP protocol header.

IAsyncSocketBufferSizeOptions Class

IAsyncSocketBufferSizeOptions Class	Description
•   SetReceiveBufferSize (see page 621)	Sets a socket level option which determines the maximum socket receive buffer size.
•   SetTransmitBufferSize (see page 621)	Sets a socket level option which determines the maximum socket transmit buffer size.

CEComUnknown Class

CEComUnknown Class	Description
•   InitializeInstance (see page 416)	Initializes the instance.

Legend

	Method
	virtual
	abstract

2.11.1.10.1 - Methods

2.11.1.10.1.1 - CTcpSocketOptions::ApplyOptions Method

Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

C++

```
virtual mxt_result ApplyOptions(IN IAsyncSocket* pIAsyncSocket);
```

Returns

resS_OK: The socket options have been successfully set. resFE_MITOSFW_ECOM_NOINTERFACE

Description

Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

2.11.1.10.1.2 - CTcpSocketOptions::CreateInstance Method

This is the ECOM (see page 412) Creation handler.

C++

```
static mxt_result CreateInstance(IN IEComUnknown* pOuterIEComUnknown, OUT CEComUnknown** ppCEComUnknown);
```

2.11.1.10.1.3 - CTcpSocketOptions::NonDelegatingQueryIf Method

The following statement is essential, it provide the default and unique implementation of the IEComUnknown (see page 416) interface that every other interfaces inherit from.

C++

```
virtual MX_DECLARE_DELEGATING_IECOMUNKNOWN mxt_result NonDelegatingQueryIf(IN mxt_iid iidRequested, OUT void** ppInterface);
```

2.11.1.10.1.4 - CTcpSocketOptions::Set8021QUserPriority Method

Sets a socket level option which enables/disables utilization of the 802.1Q user priority.

C++

```
virtual mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority);
```

Parameters

Parameters	Description
bEnable	Determines whether the option is enabled or not.
uUserPriority	The value to use in the 802.1Q user priority field.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets a socket level option which enables/disables utilization of the 802.1Q user priority. If the option is enabled, then the 802.1Q user priority field is set to uUserPriority. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.10.1.5 - CTcpSocketOptions::SetConnectTimeoutMs Method

Sets the connect timeout in ms.

C++

```
virtual void SetConnectTimeoutMs(IN uint64_t uConnectTimeoutMs);
```

Parameters

Parameters	Description
uConnectTimeoutMs	The connect timeout.

Description

Sets the connect timeout in ms. If a TCP socket is not connected when the timeout occurs, an error is reported to the manager. If the OS timer is smaller than this value, it has priority.

2.11.1.10.1.6 - CTcpSocketOptions::SetKeepAlive Method

Enables/disables the TCP keep-alive option.

C++

```
virtual mxt_result SetKeepAlive(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	If true, enables the TCP keep-alive messages on the socket.

Returns

resS_OK: The option has been successfully enabled. See GetSocketErrorId for possible return values.

Description

This method is used to enable/disable the TCP socket option to send keep-alive messages on an idle TCP connection, using the operating system's default settings.

2.11.1.10.1.7 - **CTcpSocketOptions::SetNagle** Method

Enables/disables the TCP Nagle algorithm option.

C++

```
virtual mxt_result SetNagle(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	If true, enables the TCP Nagle algorithm on the socket.

Returns

resS_OK: The option has been successfully enabled. See GetSocketErrorId for possible return values.

Description

This method is used to enable/disable the TCP Nagle algorithm option. The Nagle algorithm controls how the TCP stream is packed into TCP fragment.

2.11.1.10.1.8 - **CTcpSocketOptions::SetReceiveBufferSize** Method

Sets a socket level option which determines the maximum socket receive buffer size.

C++

```
virtual mxt_result SetReceiveBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the receive buffer.

Returns

resS_OK: The receive buffer size has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets a socket level option which determines the maximum socket receive buffer size in bytes. Usually the default value of this option is adequate; only experts should change this value. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.10.1.9 - **CTcpSocketOptions::SetTos** Method

Sets a socket level option which sets the TOS byte field in the IP protocol header.

C++

```
virtual mxt_result SetTos(IN uint8_t uTos);
```

Parameters

Parameters	Description
uTos	The value of the TOS byte field for this socket.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets a socket level option which sets the TOS byte field in the IP protocol header. This method can be called even if the socket is not bound yet.

Windows platforms specific information:

For this option to work under Windows 2000, XP, and 2003 Server, the value **DisableUserTOSSetting** must be set to 0 in the registry.

It is located in the following key: HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters

The value is a REG_DWORD that must be set to 0 in order for the TOS flags to be set in the IP header. On Windows 2000 and some older builds of Windows XP, the value must be added. Windows must be rebooted for the changes to take effect. Additional information can be found here: <http://support.microsoft.com/default.aspx?scid=kb;en-us;248611>

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Windows CE), the execution of this method will output an error trace before returning with resS_OK.

In Windows CE 4.0, the function call succeeds but the TOS byte field in the IP protocol header will remain unchanged.

Under Nucleus, this method sets the DSCP bytefield instead of the TOS. It is impossible to set the TOS byte field.

2.11.1.10.1.10 - CTcpSocketOptions::SetTransmitBufferSize Method

Sets a socket level option which determines the maximum socket transmit buffer size.

C++

```
virtual mxt_result SetTransmitBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the transmit buffer.

Returns

resS_OK: The transmit buffer size has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets a socket level option which determines the maximum socket transmit buffer size in bytes. Usually the default value of this option is adequate; only experts should change this value. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.10.1.11 - CTcpSocketOptions::SetWindowsReceivingFlowspec Method

Sets the receiving flowspec (Windows only).

C++

```
virtual mxt_result SetWindowsReceivingFlowspec(IN FLOWSPEC* pReceivingFlowspec);
```

Parameters

Parameters	Description
pReceivingFlowspec	The value of the receiving flowspec.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets the receiving flowspec (Windows only). This method can be called even if the socket is not bound yet.

2.11.1.10.1.12 - CTcpSocketOptions::SetWindowsSendingFlowspec Method

Sets the sending flowspec (Windows only).

C++

```
virtual mxt_result SetWindowsSendingFlowspec(IN FLOWSPEC* pSendingFlowspec);
```

Parameters

Parameters	Description
pSendingFlowspec	The value of the sending flow spec.

Returns

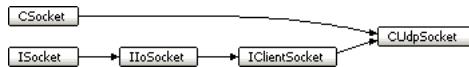
resS_OK: The option has been successfully set. See GetSocketErrorId for possible return values.

Description

This method sets the sending flowspec (Windows only). This method can be called even if the socket is not bound yet.

2.11.1.11 - CUdpSocket Class

Implementation of a client UDP socket.

Class Hierarchy**C++**

```
class CUdpSocket : protected CSocket, public IClientSocket;
```

Description

The CUdpSocket class is an implementation of a UDP socket in client mode which supports the IClientSocket (see page 635) interface.

Location

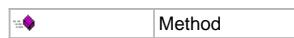
Network/CUdpSocket.h

See Also

CSomethingElse, CSomething, CMachin

Constructors

Constructor	Description
CUdpSocket (see page 586)	Default Constructor.

Legend**Methods**

Method	Description
Bind (see page 586)	Binds the socket to a local address.
Close (see page 586)	Closes a socket.
Connect (see page 587)	Connects the socket to a remote address.
Create (see page 587)	Create the socket.
GetAddressFamily (see page 588)	Retrieves the address family of this socket.
GetHandle (see page 588)	Retrieve the socket handle.
GetLocalAddress (see page 588)	Retrieves the local address to which the socket is bound.
GetPeerAddress (see page 588)	Retrieves the remote address to which the socket is connected.
GetProtocolFamily (see page 589)	Retrieves the protocol family of this socket.
GetSocketType (see page 589)	Retrieves information about the socket and its transport.
Recv (see page 589)	Receives data from a connected socket.
RecvFrom (see page 590)	Receives data from a non-connected socket.
Release (see page 591)	Deletes this socket.
Send (see page 592)	Sends data on a connected socket.
SendTo (see page 593)	Sends data on a non-connected socket.
Set8021QUserPriority (see page 594)	Set 8021Q user priority option.
SetAllowAnySource (see page 594)	Set allow any source option.
SetBlocking (see page 594)	Set socket operation to blocking/non-blocking
SetBroadcast (see page 595)	Set socket broadcast option
SetIpv6UnicastHops (see page 595)	Sets the IPv6 outgoing unicast hop limit.
SetReceiveBufferSize (see page 595)	Set socket receive buffer size option
SetTos (see page 596)	Set socket type of service (TOS byte) option
SetTransmitBufferSize (see page 596)	Set socket transmit buffer size option
SetUdpChecksum (see page 596)	Enable / Disable computation of UDP checksum

• SetWindowsReceivingFlowspec (see page 596)	Sets the receiving flowspec in windows only.
• SetWindowsSendingFlowspec (see page 597)	Sets the sending flowspec in windows only.

IClientSocket Class

IClientSocket Class	Description
• Bind (see page 636)	Binds the socket to a local address.
• Connect (see page 637)	Connects the socket to a remote address.

Legend

	Method
	virtual
	abstract

2.11.1.11.1 - Constructors

2.11.1.11.1.1 - CUdpSocket::CUdpSocket Constructor

Default Constructor.

C++

```
CUdpSocket();
```

Description

Constructor.

2.11.1.11.2 - Methods

2.11.1.11.2.1 - CUdpSocket::Bind Method

Binds the socket to a local address.

C++

```
virtual mxt_result Bind(IN const CSocketAddr* pLocalAddress, OUT CSocketAddr* pEffectiveLocalAddress);
```

Parameters

Parameters	Description
pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.
pEffectiveLocalAddress	Upon return, contains the address to which the socket has effectively been bound. It may be NULL if the effective binding is not important to the caller.

Returns

- resS_OK: Socket has been bound.
- Failure codes: Socket has not been bound.

Description

Used to associate a local address to the socket.

2.11.1.11.2.2 - CUdpSocket::Close Method

Closes a socket.

C++

```
virtual mxt_result Close(IN ECloseBehavior eBehavior);
```

Parameters

Parameters	Description
eBehavior	How the socket is to be closed, as defined by the ECloseBehavior enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to close a socket. This puts the socket into the state it was just after its allocation. After this call, all system resources associated to the socket must have been released. Note that a socket can be reused after it has been closed. In such a case, the socket options are all back to their default values. In other words, calling Close() on a socket has the effect of resetting the socket options to their default values after it is closed.

2.11.1.11.2.3 - CUdpSocket::Connect Method

Connects the socket to a remote address.

C++

```
virtual mxt_result Connect(IN const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pPeerAddress	Remote address where the socket is to be connected.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to connect the socket to a remote address. Once a socket is connected, Send and Recv can be used to exchange data with the remote peer.

2.11.1.11.2.4 - CUdpSocket::Create Method

Create the socket.

C++

```
mxt_result Create(IN CSocketAddr::EAddressFamily eAddressFamily);
```

Parameters

Parameters	Description
IN CSocketAddr::EAddressFamily eAddressFamily	The address family to use.

Returns

- resS_OK: Method processed successfully.
- resFE_INVALID_STATE: Already created.
- resFE_NOT_IMPLEMENTED: Protocol family not supported.
- resFE_FAIL: Socket failed to be created.
- See GetSocketErrorId (see page 656) for possible return values.
- Results returned by WSAlioctl_SIO_UDP_CONNRESET.

Description

This method is used to create a socket to communicate over the UDP protocol using the protocol family corresponding to the provided address family. This will only allocate all system resources required for the socket to communicate. Any other methods called before Create() will result in an error.

2.11.1.11.2.5 - CUdpSocket::GetAddressFamily Method

Retrieves the address family of this socket.

C++

```
virtual mxt_result GetAddressFamily(OUT CSocketAddr::EAddressFamily* peAddressFamily) const;
```

Parameters

Parameters	Description
peAddressFamily	OUT parameter that receives the address family, as defined in the CSocketAddr::EAddressFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address family associated to the socket.

2.11.1.11.2.6 - CUdpSocket::GetHandle Method

Retrieve the socket handle.

C++

```
mxt_hSocket GetHandle() const;
```

Returns

The socket's handle.

Description

This method is used to retrieve the socket handle.

2.11.1.11.2.7 - CUdpSocket::GetLocalAddress Method

Retrieves the local address to which the socket is bound.

C++

```
virtual mxt_result GetLocalAddress(OUT CSocketAddr* pLocalAddress) const;
```

Parameters

Parameters	Description
pLocalAddress	OUT parameter that will contain the address and port information.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address and port to which the socket has been bound. It will return an error until the socket is bound to an interface with the Bind function.

2.11.1.11.2.8 - CUdpSocket::GetPeerAddress Method

Retrieves the remote address to which the socket is connected.

C++

```
virtual mxt_result GetPeerAddress(OUT CSocketAddr* pPeerAddress) const;
```

Parameters

Parameters	Description
pPeerAddress	OUT parameter used to receive the address.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address to which the socket is currently connected. It can return an error for unconnected sockets.

2.11.1.11.2.9 - CUdpSocket::GetProtocolFamily Method Deprecated since v2.1.4

Retrieves the protocol family of this socket.

C++

```
virtual mxt_result GetProtocolFamily(OUT EProtocolFamily* peProtocolFamily) const;
```

Parameters

Parameters	Description
peProtocolFamily	OUT parameter that receives the protocol family, as defined in the EProtocolFamily enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the protocol family associated to the socket. This method is deprecated and GetAddressFamily should be used instead.

2.11.1.11.2.10 - CUdpSocket::GetSocketType Method

Retrieves information about the socket and its transport.

C++

```
virtual mxt_result GetSocketType(OUT ESocketType* peSocketType) const;
```

Parameters

Parameters	Description
peSocketType	OUT parameter that receives the socket type information, as defined in the ESocketType enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve information about this socket and its transport.

2.11.1.11.2.11 - Recv**2.11.1.11.2.11.1 - CUdpSocket::Recv Method**

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT CBlob* pData);
```

Parameters

Parameters	Description
pData	Blob object that will be filled with the data received on the socket.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from a connected socket.

The maximum number of bytes that can be received is defined by the blob's capacity. The user of the socket might want to resize the blob to a specific size before calling Recv, as its capacity will not be changed by this call.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::Recv
- IloSocket::RecvFrom

2.11.1.11.2.11.2 - CUdpSocket::Recv Method

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize);
```

Parameters

Parameters	Description
puData	Pointer to a buffer where the received data is to be stored.
uCapacity	The maximum size available in the buffer pointed to by puData.
puSize	OUT parameter that contains the number of bytes actually written in the buffer.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from a connected socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::Recv
- IloSocket::RecvFrom

2.11.1.11.2.12 - RecvFrom**2.11.1.11.2.12.1 - CUdpSocket::RecvFrom Method**

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT CBlob* pData, OUT CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pData	Blob object that will be filled with the data received on the socket.
pPeerAddress	Address from which the data was received.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::RecvFrom
- IloSocket::Recv

2.11.1.11.2.12.2 - CUdpSocket::RecvFrom Method

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize, OUT CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
puData	Pointer to a buffer where the received data is to be stored.
uCapacity	The maximum size available in the buffer pointed to by puData.
puSize	OUT parameter that contains the number of bytes actually written in the buffer.
pPeerAddress	Address from which the data was received.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- IloSocket::RecvFrom
- IloSocket::Recv

2.11.1.11.2.13 - CUdpSocket::Release Method

Deletes this socket.

C++

```
virtual uint32_t Release();
```

Returns

Always zero.

Description

This method is used to delete a socket. This has become necessary in order to be able to delete the socket from one of its interfaces (instead of directly calling the class that implements these interfaces). This is called by the user of the socket instead of the delete operator.

Note that this can be called if the socket is not closed yet. In such a case, the socket will be closed automatically before the socket is deleted.

The return value is always zero. We eventually plan to implement reference counting on the sockets, and the reference count number will be returned instead of zero when calling Release.

2.11.1.11.2.14 - Send

2.11.1.11.2.14.1 - CUdpSocket::Send Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const CBlob* pData, OUT unsigned int* puSizeSent);
```

Parameters

Parameters	Description
pData	Pointer to a blob object that contains the data to be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::Send
- IloSocket::SendTo

2.11.1.11.2.14.2 - CUdpSocket::Send Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent);
```

Parameters

Parameters	Description
puData	Pointer to a buffer containing the data to send.
uSize	The number of bytes in the buffer that must be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::Send
- IloSocket::SendTo

2.11.1.11.2.15 - SendTo**2.11.1.11.2.15.1 - CUdpSocket::SendTo Method**

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const CBlob* pData, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
pData	Pointer to a blob containing the data to be sent.
puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
pPeerAddress	Address to which the data must be sent.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::SendTo
- IloSocket::Send

2.11.1.11.2.15.2 - CUdpSocket::SendTo Method

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress);
```

Parameters

Parameters	Description
puData	Pointer to a buffer containing the data to send.
uSize	The number of bytes in the buffer that must be sent.

puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
pPeerAddress	Address to which the data must be sent.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::SendTo
- IloSocket::Send

2.11.1.11.2.16 - CUdpSocket::Set8021QUserPriority Method

Set 8021Q user priority option.

C++

```
mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority);
```

Parameters

Parameters	Description
bEnable	Determine whether the option is enable or not.
uUserPriority	The value to use in the 8021Q user priority field.

Returns

- resS_OK: The value has been correctly set.
- results returned by SetSockOpt8021QUserPriority or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable reprenting the current state of the option.

2.11.1.11.2.17 - CUdpSocket::SetAllowAnySource Method

Set allow any source option.

C++

```
mxt_result SetAllowAnySource(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the option is enable (true) or not (false).

Returns

- Results returned by SetSockOptAllowAnySource.

Description

Simply redirect the call to a generic implementation and update internal variable reprenting the current state of the option.

2.11.1.11.2.18 - CUdpSocket::SetBlocking Method

Set socket operation to blocking/non-blocking

C++

```
mxt_result SetBlocking(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the socket operation mode is blocking (true) or not (false).

Returns

- Results returned by SetSockOptBlocking.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.11.2.19 - CUdpSocket::SetBroadcast Method

Set socket broadcast option

C++

```
mxt_result SetBroadcast(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Determine if the socket will send broadcast or not. When bEnable is true broadcast are sent otherwise, unicast are sent.

Returns

- Results returned by SetSockOptBroadcast.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

Notes

This option has no effect on IPv6 sockets.

2.11.1.11.2.20 - CUdpSocket::SetIpv6UnicastHops Method

Sets the IPv6 outgoing unicast hop limit.

C++

```
mxt_result SetIpv6UnicastHops(IN int nHops);
```

Parameters

Parameters	Description
nHops	Number of hops.

Returns

- Results returned by SetSockOptIpv6UnicastHops.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.11.2.21 - CUdpSocket::SetReceiveBufferSize Method

Set socket receive buffer size option

C++

```
mxt_result SetReceiveBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the receive buffer.

Returns

- Results returned by SetSockOptReceiveBufferSize.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option. Usually, the default value of this option is adequate. Do not modify it unless you know what you do.

2.11.1.11.2.22 - CUdpSocket::SetTos Method

Set socket type of service (TOS byte) option

C++

```
mxt_result SetTos(IN uint8_t uTos);
```

Parameters

Parameters	Description
uTos	The value of the TOS byte field for this socket.

Returns

- resS_OK
- Results returned by SetSockOptTos or UpdateQoSSettings.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option.

2.11.1.11.2.23 - CUdpSocket::SetTransmitBufferSize Method

Set socket transmit buffer size option

C++

```
mxt_result SetTransmitBufferSize(IN unsigned int uSize);
```

Parameters

Parameters	Description
uSize	The size of the transmit buffer.

Returns

- Results returned by SetSockOptTransmitBufferSize.

Description

Simply redirect the call to a generic implementation and update internal variable representing the current state of the option. Usually, the default value of this option is adequate. Do not modify it unless you know what you do.

2.11.1.11.2.24 - CUdpSocket::SetUdpChecksum Method

Enable / Disable computation of UDP checksum

C++

```
mxt_result SetUdpChecksum(IN bool bEnable);
```

Parameters

Parameters	Description
bEnable	Indicates if the UDP checksum is enabled (true) or disabled (false).

Returns

- Results returned by SetSockOptUdpChecksum.

Description

Simply redirect the call to a generic implementation and update the internal variable representing the current state of the option.

2.11.1.11.2.25 - CUdpSocket::SetWindowsReceivingFlowspec Method

Sets the receiving flowspec in windows only.

C++

```
mxt_result SetWindowsReceivingFlowspec( IN FLOWSPEC* pReceivingFlowspec);
```

Parameters

Parameters	Description
pReceivingFlowspec	Pointer to the receiving FLOWSPEC to set.

Returns

- resS_OK

Description

Sets the receiving flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.11.2.26 - CUdpSocket::SetWindowsSendingFlowspec Method

Sets the sending flowspec in windows only.

C++

```
mxt_result SetWindowsSendingFlowspec( IN FLOWSPEC* pSendingFlowspec);
```

Parameters

Parameters	Description
pSendingFlowspec	Pointer to the sending FLOWSPEC to set.

Returns

- resS_OK

Description

Sets the sending flowspec for the socket. This method is only available under windows when the MXD_OS_WINDOWS_ENABLE_GQOS_QOS flag is enabled. The flowspec will be applied to the socket on the next call to Recv() or Send() in case of a client socket, and Listen() in case of a server socket.

2.11.1.12 - CUdpTracing Class

UDP trace output handler.

Class Hierarchy**C++**

```
class CUdpTracing : private IAsyncResolverUser;
```

Description

This static class is used to add an output handler to the tracing mechanism. This output handler sends traces as clear text on a network using a UDP socket.

Notes

For the UDP output handler to work, you must define MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS (see page 310) with a value of at least 2, in your PreMxConfig.h (see page 317).

Location

Network/CUdpTracing.h

See Also

MxTrace, CUdpSocket (see page 585)

Methods

Method	Description
◆ Finalize (see page 598)	Closes the socket and unregisters the output handler.

 Initialize (see page 598)	Initializes the socket and registers the output handler.
---	--

IAsyncResolverUser Class

IAsyncResolverUser Class	Description
  EvAsyncResolverUserResponseReceived (see page 760)	Notification of the reception of a response to a NAPTR (ENUM) question.

Legend

	Method
	abstract

2.11.1.12.1 - Methods

2.11.1.12.1.1 - CUdpTracing::Finalize Method

Closes the socket and unregisters the output handler.

C++

```
static mxt_result Finalize();
```

Returns

- resS_OK: The trace output handler was unregistered successfully and the socket was closed.
- resFE_FAIL: A call to Finalize was made without a call to Initialize (see page 598) first.

Description

This method is used to unregister the output handler from the tracing mechanism and to close the socket. This method must be called after a corresponding call to Initialize (see page 598).

See Also

Initialize (see page 598)

2.11.1.12.1.2 - CUdpTracing::Initialize Method

Initializes the socket and registers the output handler.

C++

```
static mxt_result Initialize(IN const CFqdn* pRemoteFqdn, IN const CSocketAddr* pLocalAddr = NULL, IN unsigned int uFqdnValidationPeriodMs = 30000);
```

Parameters

Parameters	Description
IN const CFqdn* pRemoteFqdn	A pointer to a CFqdn (see page 527) object containing the address and port of the destination to which we want to send the traces. This parameter must not be NULL.
IN const CSocketAddr* pLocalAddr = NULL	A pointer to a CSocketAddr (see page 545) object containing the address and port to use to send the traces on the network. This parameter can be NULL.
IN unsigned int uFqdnValidationPeriodMs = 30000	The period in mSec that pRemoteFqdn resolved address is valid. After this period, a new resolution will take place automatically.

Returns

- resS_OK: The initialization finished without errors.
- resFE_INVALID_ARGUMENT: Either the pRemoteFqdn passed as a parameter is NULL, or one of the supplied addresses is invalid.

Description

This method is used to register the UDP trace output handler and initializes the socket to be used by the output handler. This method must be called before sending traces that we want to send via UDP. This method can be called more than once to change the destination address. So far, only IPv4 is supported.

See Also

Finalize (see page 598)

2.11.1.13 - GetSockOptError Function

Gets the error status of the socket.

C++

```
mxt_result GetSockOptError(IN mxt_hSocket hSocket, OUT unsigned int* puError);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to get the error
OUT unsigned int* puError	Pointer to contain the error status of the socket.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

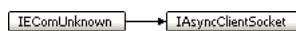
Description

Retrieves the error status of a socket.

2.11.1.14 - IAsyncClientSocket Class

Interface defining the methods associated with a client socket.

Class Hierarchy



C++

```
class IAsyncClientSocket : public IECOMUnknown;
```

Description

The IAsyncClientSocket interface defines the methods associated with a client socket. A client socket can be used to send and receive data, as defined in the IAsyncSocket (see page 602) interface, and it can also connect to a remote address.

This interface defines the BindA (see page 600)() and ConnectA (see page 600)() methods which are used to bind and connect a socket to a remote address, respectively. Note that some types of sockets must be connected in order to send and receive data (connection type eSTREAM and eSEQPACKET), while others can be used unconnected (connection type eDATAGRAM).

Events related to this interface are reported through the IAsyncClientSocketMgr (see page 601) interface.

Location

Network/IAsyncClientSocket.h

See Also

IAsyncClientSocketMgr (see page 601) IAsyncSocket (see page 602) IAsyncServerSocket (see page 607) IAsyncSocket (see page 612) IAsyncUnconnectedIoSocket (see page 631)

Methods

Method	Description
BindA (see page 600)	Binds the socket to a local address.
ConnectA (see page 600)	Connects the socket to a remote address.
SetAsyncClientSocketMgr (see page 600)	Sets the manager associated with the interface IAsyncClientSocket

IECOMUnknown Class

IECOMUnknown Class	Description
AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
QueryIf (see page 418)	Queries an object for a supported interface.
ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

Method	
abstract	

2.11.1.14.1 - Methods

2.11.1.14.1.1 - IAsyncClientSocket::BindA Method

Binds the socket to a local address.

C++

```
virtual mxt_result BindA(IN const CSocketAddr* pLocalAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.

Returns

resFE_INVALID_STATE: The socket is not in a valid state to bind on a specific address. See GetSocketErrorId (see page 656) for possible return values.

Description

Used to associate a local address to the socket. The user MUST call this method following a call to CAsyncSocketFactory::CreateAsyncSocket (see page 524)().

Upon a successful bind, the socket will generate the IAsyncClientSocketMgr::EvAsyncClientSocketMgrBound (see page 601) event. You will then be able to query the real address and port used by the socket by calling method IAsyncSocket::GetLocalAddress (see page 615) on it.

Upon a binding error, the socket will generate the IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) event.

See Also

IAsyncClientSocketMgr.h

2.11.1.14.1.2 - IAsyncClientSocket::ConnectA Method

Connects the socket to a remote address.

C++

```
virtual mxt_result ConnectA(IN const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pPeerAddress	Remote address where the socket is to be connected. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is not in a valid state to connect to a remote host. resFE_INVALID_ARGUMENT: pPeerAddress is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This asynchronous method is used to connect the socket to a remote address. Once a socket is connected, IAsyncSocket::Send (see page 604) and IAsyncSocket::Recv (see page 603) can be used to exchange data with the remote peer.

Upon a successful connection, the socket will generate the IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected (see page 602) event.

Upon a connection error, the socket will generate the IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) event.

See Also

IAsyncClientSocketMgr.h

2.11.1.14.1.3 - IAsyncClientSocket::SetAsyncClientSocketMgr Method

Sets the manager associated with the interface IAsyncClientSocket (see page 599)

C++

```
virtual mxt_result SetAsyncClientSocketMgr(IN IAsyncClientSocketMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncClientSocketMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

resFE_INVALID_STATE: The manager is probably already set. resFE_INVALID_ARGUMENT: pMgr is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets the manager associated with the interface IAsyncClientSocket (see page 599).

2.11.1.15 - IAsyncClientSocketMgr Class

Interface through which asynchronous client sockets report their events.

Class Hierarchy

IAsyncClientSocketMgr

C++

```
class IAsyncClientSocketMgr;
```

Description

This is the interface through which the asynchronous client sockets report their events. All events are reported asynchronously with respect to the manager's execution context.

Location

Network/IAsyncClientSocketMgr.h

See Also

IAsyncClientSocket (see page 599)

Methods

Method	Description
◆ A EvAsyncClientSocketMgrBound (see page 601)	Notifies that the client socket has been bound.
◆ A EvAsyncClientSocketMgrConnected (see page 602)	Notifies of a successful connection to a remote server.

Legend

◆	Method
A	abstract

2.11.1.15.1 - Methods**2.11.1.15.1.1 - IAsyncClientSocketMgr::EvAsyncClientSocketMgrBound Method**

Notifies that the client socket has been bound.

C++

```
virtual void EvAsyncClientSocketMgrBound(IN mxt_opaque opq, IN CSocketAddr* pEffectiveLocalAddress) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket that was being bound.
IN CSocketAddr* pEffectiveLocalAddress	Effective local address where the socket is bound.

Description

This is the event generated by the asynchronous client socket upon being bound.

Unsuccessful binding attempts are reported through the EvAsyncSocketMgrErrorDetected event of the IAsyncSocketMgr (see page

624) interface.

See Also

[IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected](#) (see page 625)

2.11.1.15.1.2 - **IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected** Method

Notifies of a successful connection to a remote server.

C++

```
virtual void EvAsyncClientSocketConnected(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket that was connecting.

Description

This is the event generated by the asynchronous client socket that is connecting to a remote server. This event is generated upon a successful completion of the connection.

Unsuccessful connection attempts are reported through the [EvAsyncSocketMgrErrorDetected](#) event of the [IAsyncSocketMgr](#) (see page 624) interface.

See Also

[IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected](#) (see page 625)

2.11.1.16 - **IAsyncIoSocket** Class

Interface defining the methods accessible on connected asynchronous sockets to send and receive data.

Class Hierarchy



C++

```
class IAsyncIoSocket : public IEComUnknown;
```

Description

The [IAsyncIoSocket](#) interface defines the methods accessible on a connected asynchronous socket that can be used to send and receive data.

Events related to this interface are reported through the [IAsyncIoSocketMgr](#) (see page 605) interface:

- The event [IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToRecv](#) (see page 606) is generated

when there is data ready for reception on the socket. This means that calling [IAsyncIoSocket::Recv](#) (see page 603) should yield incoming data. [IAsyncIoSocket::Recv](#) (see page 603) must be called repeatedly until it returns with zero bytes received. If this condition is not reached, the event will never be generated again.

- The event [IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToSend](#) (see page 606) is generated

once the socket becomes writable after a call to [IAsyncIoSocket::Send](#) (see page 604) has returned successfully but with less than the provided buffer size bytes sent. This means that sending of buffer of 10 bytes from which 0 to 9 bytes were sent will generate the event to notify the application that the socket is ready to send new data. At that point the application can send the remaining 1 byte.

Although the interface name seems to specify that the [IAsyncIoSocket::Recv](#) (see page 603) and [IAsyncIoSocket::Send](#) (see page 604) methods are asynchronous, this is not the case. These methods are synchronous because the socket should not queue data. Queueing data in the socket could end up with a non deterministic message queue allocation. The asynchronous part is managed through the asynchronous notification of the [IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToRecv](#) (see page 606) and [IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToSend](#) (see page 606). It is guaranteed that these two events will never be reported to the manager while a send or receive operation is in progress.

Location

Network/IAsyncIoSocket.h

See Also

IAsyncClientSocket (see page 599) IAsyncIoSocketMgr (see page 605) IAsyncServerSocket (see page 607) IAsyncSocket (see page 612) IAsyncUnconnectedIoSocket (see page 631)

Methods

Method	Description
• A GetPeerAddress (see page 603)	Retrieves the remote address to which the socket is connected.
• A Recv (see page 603)	Receives data from a connected socket.
• A Send (see page 604)	Sends data on a connected socket.
• A SetAsyncIoSocketMgr (see page 605)	Sets the manager associated with the interface IAsyncIoSocket

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
• A	abstract

2.11.1.16.1 - Methods**2.11.1.16.1.1 - IAsyncIoSocket::GetPeerAddress Method**

Retrieves the remote address to which the socket is connected.

C++

```
virtual mxt_result GetPeerAddress(OUT CSocketAddr* pPeerAddress) const = 0;
```

Parameters

Parameters	Description
OUT CSocketAddr* pPeerAddress	OUT (see page 39) parameter used to receive the address. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is not connected. resFE_INVALID_ARGUMENT: pPeerAddress is NULL. resS_OK: pPeerAddress holds the peer information.

Description

This method is used to retrieve the address to which the socket is currently connected. It can return an error for unconnected sockets.

2.11.1.16.1.2 - Recv**2.11.1.16.1.2.1 - IAsyncIoSocket::Recv Method**

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT CBlob* pData) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pData	Blob object that will be filled with the data received on the socket. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is not connected. resFE_INVALID_ARGUMENT: pData is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to receive data from a connected socket. Although this method is usually called when the socket user is notified that there is data available on the socket, it can also be called without having received such a notification. The **IAsyncloSocket** (see page 602) implementation notifies its user of the presence of incoming data through the **IAsyncloSocketMgr::EvAsyncloSocketMgrReadyToRecv** (see page 606) event.

The maximum number of bytes that can be received is defined by the blob's capacity. The user of the socket might want to resize the blob to a specific size before calling **IAsyncloSocket::Recv**, as its capacity will not be changed by this call.

Note that this method is synchronous. The **IAsyncloSocket** (see page 602) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the **IAsyncloSocket** (see page 602) interface.

See Also

IAsyncUnconnectedloSocket::RecvFrom (see page 632)

2.11.1.16.1.2.2 - **IAsyncloSocket::Recv** Method

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puData	Pointer to a buffer where the received data is to be stored. It cannot be NULL.
IN unsigned int uCapacity	The maximum size available in the buffer pointed to by puData.
OUT unsigned int* puSize	OUT (see page 39) parameter that contains the number of bytes actually written in the buffer. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is not connected. **resFE_INVALID_ARGUMENT**: puData or puSize is NULL. See **GetSocketErrorId** (see page 656) for possible return values.

Description

This method is used to receive data from a connected socket. Although this method is usually called when the socket user is notified that there is data available on the socket, it can also be called without having received such a notification. The **IAsyncloSocket** (see page 602) implementation notifies its user of the presence of incoming data through the **IAsyncloSocketMgr::EvAsyncloSocketMgrReadyToRecv** (see page 606) event.

Note that this method is synchronous. The **IAsyncloSocket** (see page 602) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the **IAsyncloSocket** (see page 602) interface.

See Also

IAsyncUnconnectedloSocket::RecvFrom (see page 632)

2.11.1.16.1.3 - **Send**

2.11.1.16.1.3.1 - **IAsyncloSocket::Send** Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const CBlob* pData, OUT unsigned int* puSizeSent) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Pointer to a blob object that contains the data to be sent. It cannot be NULL.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. It cannot be NULL. The value returned in this parameter is not reliable if the method returns anything else than resS_OK .

Returns

resFE_INVALID_STATE: The socket is not connected. **resFE_INVALID_ARGUMENT**: pData or puSizeSent is NULL. See **GetSocketErrorId** (see page 656) for possible return values.

Description

This method is used to send data on a connected socket. Note that the `IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToSend` (see page 606) event will be reported to the manager if this method succeeds and returns with `*puSizeSent >= 0` and `<` to the `pData` size.

Note that this method is synchronous. The `IAsyncIoSocket` (see page 602) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncIoSocket` (see page 602) interface.

An application MUST not rely on the value of `puSizeSent` when this method returns anything else than `resS_OK`.

See Also

`IAsyncUnconnectedIoSocket::SendTo` (see page 633)

2.11.1.16.1.3.2 - `IAsyncIoSocket::Send` Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to a buffer containing the data to send. It cannot be NULL.
IN unsigned int uSize	The number of bytes in the buffer that must be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. It cannot be NULL. The value returned in this parameter is not reliable if the method returns anything else than <code>resS_OK</code> .

Returns

`resFE_INVALID_STATE`: The socket is not connected. `resFE_INVALID_ARGUMENT`: `puData` or `puSizeSent` is NULL. See `GetSocketErrorId` (see page 656) for possible return values.

Description

This method is used to send data on a connected socket. Note that the `IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToSend` (see page 606) event will be reported to the manager if this method succeeds and returns with `*puSizeSent >= 0` and `<` to the `pData` size.

Note that this method is synchronous. The `IAsyncIoSocket` (see page 602) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncIoSocket` (see page 602) interface.

An application MUST not rely on the value of `puSizeSent` when this method returns anything else than `resS_OK`.

See Also

`IAsyncUnconnectedIoSocket::SendTo` (see page 633)

2.11.1.16.1.4 - `IAsyncIoSocket::SetAsyncIoSocketMgr` Method

Sets the manager associated with the interface `IAsyncIoSocket` (see page 602)

C++

```
virtual mxt_result SetAsyncIoSocketMgr(IN IAsyncIoSocketMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncIoSocketMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

`resFE_INVALID_STATE`: The manager is probably already set. `resFE_INVALID_ARGUMENT`: `pMgr` is NULL. See `GetSocketErrorId` (see page 656) for possible return values.

Description

This method sets the manager associated with the interface `IAsyncIoSocket` (see page 602).

2.11.1.17 - `IAsyncIoSocketMgr` Class

This is the interface through which the asynchronous I/O sockets report their events.

Class Hierarchy

IAsyncIoSocketMgr

C++

```
class IAsyncIoSocketMgr;
```

Description

This is the interface through which the asynchronous I/O sockets report their events. All events are reported asynchronously with respect to the manager's execution context.

Location

Network/IAsyncIoSocketMgr.h

See Also

IAsyncIoSocket (see page 602)

Methods

Method	Description
◆ A EvAsyncIoSocketMgrReadyToRecv (see page 606)	Notifies the socket user that there is data available to be received on the socket.
◆ A EvAsyncIoSocketMgrReadyToSend (see page 606)	Notifies the socket user that the socket is ready to send data on the network.

Legend

◆	Method
A	abstract

2.11.1.17.1 - Methods

2.11.1.17.1.1 - IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToRecv Method

Notifies the socket user that there is data available to be received on the socket.

C++

```
virtual void EvAsyncIoSocketMgrReadyToRecv(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque parameter associated to the socket.

Description

This event is reported by a socket when there is data available on the socket for reading. This means that the socket user can call IAsyncIoSocket::Recv (see page 603) or IAsyncUnconnectedIoSocket::RecvFrom (see page 632) at least once without blocking on the socket.

Note that there might be some instances of sockets (depending on the protocol) that do generate this event with a call to Recv or RecvFrom yielding zero bytes read. Thus, the socket user should be ready to handle this case.

See Also

IAsyncIoSocket::Recv@CBlob (see page 95)*, IAsyncIoSocket::Recv@uint8_t (see page 85)*, unsigned int, unsigned int*, IAsyncIoSocket::RecvFrom@CBlob (see page 95)*, CSocketAddr (see page 545)*, IAsyncIoSocket::RecvFrom@uint8_t (see page 85)*, unsigned int, unsigned int*, CSocketAddr (see page 545)*,

2.11.1.17.1.2 - IAsyncIoSocketMgr::EvAsyncIoSocketMgrReadyToSend Method

Notifies the socket user that the socket is ready to send data on the network.

C++

```
virtual void EvAsyncIoSocketMgrReadyToSend(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque parameter associated to the socket.

Description

This event is reported by a socket when it is possible for its user to send data on the network. Thus, after receiving this, the socket user should be able to use the various flavors of `IAsyncSocket::Send` (see page 604) and `IAsyncUnconnectedSocket::SendTo` (see page 633) to send the data on the network.

Note that there might be some instances of sockets (depending on the protocol) that do generate this event even if a call to `IAsyncSocket::Send` (see page 604) and `IAsyncUnconnectedSocket::SendTo` (see page 633) will fail with `eFE_SOCK_WOULDBLOCK` and zero bytes written. The socket user must be able to handle this case.

See Also

`IAsyncSocket::Send@CBlob` (see page 95)* `IAsyncSocket::Send@uint8_t` (see page 85)*, `unsigned int`
`IAsyncSocket::SendTo@CBlob` (see page 95)*, `CSocketAddr` (see page 545)* `IAsyncSocket::SendTo@uint8_t` (see page 85)*,
`unsigned int`, `CSocketAddr` (see page 545)*

2.11.1.18 - IAsyncServerSocket Class

The `IServerSocket` (see page 644) interface defines the methods needed for a server type of socket.

Class Hierarchy



C++

```
class IAsyncServerSocket : public IEComUnknown;
```

Description

The `IAsyncServerSocket` interface defines the methods needed for a server type of socket. Note that this interface is available only to connection-oriented sockets types.

Events related to this interface are reported through the `IAsyncServerSocketMgr` (see page 610) interface.

Location

Network/IAsyncServerSocket.h

See Also

`IAsyncClientSocket` (see page 599) `IAsyncSocket` (see page 602) `IAsyncServerSocketMgr` (see page 610) `IAsyncSocket` (see page 612) `IAsyncUnconnectedSocket` (see page 631)

Methods

Method	Description
• A AcceptA (see page 607)	Accepts an incoming connection attempt.
• A BindA (see page 608)	Binds the socket to a local address.
• A Listen (see page 609)	Lists on the socket for incoming connection attempts.
• A QueryAcceptedOptionsIf (see page 609)	Provides ECOM (see page 412) interfaces to set options on the accepted TCP socket.
• A SetAsyncServerSocketMgr (see page 610)	Sets the manager associated with the interface <code>IAsyncServerSocket</code>

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.18.1 - Methods

2.11.1.18.1.1 - IAsyncServerSocket::AcceptA Method

Accepts an incoming connection attempt.

C++

```
virtual mxt_result AcceptA(IN IEComUnknown* pServicingThread) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pServicingThread	Pointer to the Servicing Thread to which the socket should be attached. It can be NULL, in which case the server socket's internal servicing thread will be used.

Returns

Always returns resS_OK. Errors will be reported through the IAsynSocketMgr::EvAsyncSocketMgrErrorDetected event.

Description

Accepts an incoming connection attempt on a listening connection-oriented socket.

Creating multiple sockets attached to the same Servicing Thread is a way to have multiple sockets running in a single thread.

The method is asynchronous because under some conditions, the process of accepting a new connection may take some time. The user will be advised when the process is completed by the IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted (see page 611) event.

If an important error is detected late in the accepting process, the error will be forwarded to the caller by means of an IAsynSocketMgr::EvAsyncSocketMgrErrorDetected event.

Deprecation Note:

The opaque parameter that used to be a parameter to this method is now deprecated. To set the opaque value of an accepted socket, the application must now wait for the IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted (see page 611) event and use the IAsyncSocket::SetOpaque (see page 619) method. An IAsyncloSocket (see page 602) interface is received as a parameter to the event on which the IAsyncSocket (see page 612) interface can be queried.

Furthermore, the application can also use the socket factory's IAsyncSocketFactoryConfigurationMgr (see page 621) interface instead to set the opaque value. See the interface's documentation for more details.

See Also

Listen (see page 609)

2.11.1.18.1.2 - IAsyncServerSocket::BindA Method

Binds the socket to a local address.

C++

```
virtual mxt_result BindA(IN const CSocketAddr* pLocalAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.

Returns

Always return resS_OK. Errors will be reported through the IAsynSocketMgr::EvAsyncSocketMgrErrorDetected event.

Description

Used to associate a local address to the socket. The user MUST call this method following a call to CAsyncSocketFactory::CreateAsyncSocket (see page 524)().

Upon a successful bind, the socket will generate the IAsyncServerSocketMgr::EvAsyncClientSocketMgrBound event. You will then be able to query the real address and port used by the socket by calling method IAsyncSocket::GetLocalAddress (see page 615) on it.

Upon a binding error, the socket will generate the IAsynSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) event.

See Also

IAsyncClientSocketMgr.h

2.11.1.18.1.3 - IAsyncServerSocket::Listen Method

Listens on the socket for incoming connection attempts.

C++

```
virtual mxt_result Listen(IN unsigned int uMaxPendingConnection = 5) = 0;
```

Parameters

Parameters	Description
IN unsigned int uMaxPendingConnection = 5	Maximum number of pending connections accepted.

Returns

resFE_INVALID_STATE: The socket is not correctly initialized. See GetSocketErrorId (see page 656) for possible return values

Description

Sets the socket to listen for incoming connection attempts. Once a connection attempt is detected, IAsyncServerSocket::AcceptA (see page 607) must be called in order to fully establish the connection.

Once the socket detects an incoming connection attempt, the IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionRequested (see page 612) event is reported. The socket user must then call IAsyncServerSocket::AcceptA (see page 607) in order to establish the connection.

See Also

AcceptA (see page 607), IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionRequested (see page 612)

2.11.1.18.1.4 - QueryAcceptedOptionsIf

2.11.1.18.1.4.1 - IAsyncServerSocket::QueryAcceptedOptionsIf Method

Provides ECOM (see page 412) interfaces to set options on the accepted TCP socket.

C++

```
virtual mxt_result QueryAcceptedOptionsIf(IN mxt_iid iidRequested, OUT void** ppInterface) = 0;
```

Parameters

Parameters	Description
IN mxt_iid iidRequested	The id of the ECOM (see page 412) interface requested for the accepted TCP socket.
OUT void** ppInterface	The returned interface. The caller is responsible of calling ReleaselfRef (see page 420) on the returned interface. It must not be NULL.

Returns

resFE_MITOSFW_ECOM_NOINTERFACE: The queried interface has not been found. resS_OK: The interface has been retrieved.

Description

This method provides ECOM (see page 412) interfaces to set options on the accepted TCP socket.

Warning

Proper reference count handling is required when using this method:

- All interfaces returned by this method MUST be released EXACTLY ONCE using ReleaselfRef (see page 420).
- All copies of the returned pointer MUST add a reference using AddIfRef (see page 418) and use ReleaselfRef (see page 420) on it when the copy is no longer needed

These statements are part of the basic ECOM (see page 412) rules. They are reminded here to stress out their importance regarding this method.

Following these rules will prevent memory leaks and crashes inside the asynchronous server socket.

2.11.1.18.1.4.2 - IAsyncServerSocket::QueryAcceptedOptionsIf Method

Provides ECOM (see page 412) interfaces to set options on the accepted TCP socket.

C++

```
template <class _Type> mxt_result QueryAcceptedOptionsIf(OUT _Type** ppInterface);
```

Parameters

Parameters	Description
OUT _Type** ppInterface	The returned interface. The caller is responsible of calling ReleaselfRef (see page 420) on the returned interface. It must not be NULL.

Returns

resFE_MITOSFW_ECOM_NOINTERFACE: The queried interface has not been found. resS_OK: The interface has been retrieved.

Description

This is a templated version of the original QueryAcceptedOptionsIf method. It is designed to ease code readability.

For this method to compile, the **ppInterface referenced class must contain MX_DECLARE_ECOM_GETIID (see page 424).

This method provides ECOM (see page 412) interfaces to set options on the accepted TCP socket.

Warning

Proper reference count handling is required when using this method:

- All interfaces returned by this method MUST be released EXACTLY ONCE using ReleaselfRef (see page 420).
- All copies of the returned pointer MUST add a reference using AddlfRef (see page 418) and use ReleaselfRef (see page 420) on it when the copy is no longer needed

These statements are part of the basic ECOM (see page 412) rules. They are reminded here to stress out their importance regarding this method.

Following these rules will prevent memory leaks and crashes inside the asynchronous server socket.

2.11.1.18.1.5 - IAsyncServerSocket::SetAsyncServerSocketMgr Method

Sets the manager associated with the interface IAsyncServerSocket (see page 607)

C++

```
virtual mxt_result SetAsyncServerSocketMgr(IN IAsyncServerSocketMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncServerSocketMgr* pMgr	Pointer to a IAsyncServerSocketMgr (see page 610) interface. It cannot be NULL.

Returns

resFE_INVALID_STATE: The manager has probably already been set. resFE_INVALID_ARGUMENT: pMgr is NULL. resS_OK: The manager has been correctly set.

Description

This method sets the manager associated with the IAsyncServerSocket (see page 607) interface.

2.11.1.19 - IAsyncServerSocketMgr Class

This is the interface through which server sockets report events.

Class Hierarchy

```
IAsyncServerSocketMgr
```

C++

```
class IAsyncServerSocketMgr;
```

Description

This is the interface through which server sockets report events. All events are reported asynchronously with respect to the manager's execution context.

Location

Network/IAsyncServerSocketMgr.h

See Also[IAsyncServerSocket](#) (see page 607)**Methods**

Method	Description
• A EvAsyncServerSocketMgrBound (see page 611)	Notifies that the server socket has been bound.
• A EvAsyncServerSocketMgrConnectionAccepted (see page 611)	The user accepted a connection request and now the accepting process is complete.
• A EvAsyncServerSocketMgrConnectionFailed (see page 612)	The user accepted a connection request and now the accepting process failed.
• A EvAsyncServerSocketMgrConnectionRequested (see page 612)	An incoming connection attempt was detected on a server socket.

Legend

• A	Method
A	abstract

2.11.1.19.1 - Methods**2.11.1.19.1.1 - IAsyncServerSocketMgr::EvAsyncServerSocketMgrBound Method**

Notifies that the server socket has been bound.

C++

```
virtual void EvAsyncServerSocketMgrBound(IN mxt_opaque opq, IN CSocketAddr* pEffectiveLocalAddress) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket that is being bound.
IN CSocketAddr* pEffectiveLocalAddress	Effective local address where the socket is bound.

Description

This is the event generated by the asynchronous server socket upon being bound.

Unsuccessful binding attempts are reported through the [IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected](#) (see page 625) event.

See Also[IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected](#) (see page 625)**2.11.1.19.1.2 - IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted Method**

The user accepted a connection request and now the accepting process is complete.

C++

```
virtual void EvAsyncServerSocketMgrConnectionAccepted(IN mxt_opaque opqServerSocketOpaque, IN IAsyncIoSocket* pAsyncIoSocket) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqServerSocketOpaque	Opaque value associated with the server socket on which the accept has been requested.
IN IAsyncIoSocket* pAsyncIoSocket	Pointer to an IAsyncIoSocket (see page 602) interface representing the accepted connection.

Description

This event is reported by a server socket when an incoming connection attempt has been accepted and the accepting process is complete.

Deprecation Note:

The opaque parameter for the accepted socket that used to be a parameter to this event is now deprecated. To set the opaque value of an accepted socket, the application must use the [IAsyncSocket::SetOpaque](#) (see page 619) method inside this event. An [IAsyncIoSocket](#) (see page 602) interface is received as a parameter to this event on which the [IAsyncSocket](#) (see page 612) interface can be queried.

Furthermore, the application can also use the socket factory's `IAsyncSocketFactoryConfigurationMgr` (see page 621) interface instead to set the opaque value. See the interface's documentation for more details.

See Also

`IAsyncServerSocket::AcceptA` (see page 607)

2.11.1.19.1.3 - `IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionFailed` Method

The user accepted a connection request and now the accepting process failed.

C++

```
virtual void EvAsyncServerSocketMgrConnectionFailed(IN mxt_opaque opqServerSocketOpaque, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
<code>IN mxt_opaque opqServerSocketOpaque</code>	Opaque value associated with the server socket on which the accept was requested.
<code>IN mxt_result res</code>	The error detected on the socket.

Description

This event is reported by a server socket when an incoming connection attempt has been accepted and the accepting process failed.

Deprecation Note:

The opaque parameter that used to be a parameter to this method is now deprecated. The opaque value of an accepted socket is now set only when the application receives the `IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted` (see page 611) event and uses the `IAsyncSocket::SetOpaque` (see page 619) method or the socket factory' `IAsyncSocketFactoryConfigurationMgr` (see page 621) interface. This event being exclusive with the `IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted` (see page 611) event, it no longer has access to the opaque value of the accepted socket nor does it need that opaque value.

See Also

`IAsyncServerSocket::AcceptA` (see page 607)

2.11.1.19.1.4 - `IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionRequested` Method

An incoming connection attempt was detected on a server socket.

C++

```
virtual void EvAsyncServerSocketMgrConnectionRequested(IN mxt_opaque opqServerSocketOpaque) = 0;
```

Parameters

Parameters	Description
<code>IN mxt_opaque opqServerSocketOpaque</code>	Opaque parameter associated with the server socket.

Description

This event is reported by a server socket when an incoming connection attempt is detected. In order to correctly establish the incoming connection, `IAsyncServerSocket::AcceptA` (see page 607) must be called on the server socket.

See Also

`IAsyncServerSocket::AcceptA` (see page 607)

2.11.1.20 - `IAsyncSocket` Class

Interface defining the basic methods accessible on asynchronous sockets.

Class implementing abstraction of sockets. It is used to encapsulate the socket functionality in an asynchronous manner.

Class Hierarchy



C++

```
class IAsyncSocket : public IEComUnknown;
```

Description

Interface defining the most basic methods accessible on asynchronous sockets. This means the asynchronous closure of the socket and retrieval of its various associated properties. All asynchronous sockets implement this interface.

Although it is possible to close an asynchronous socket by releasing all references to it, it is a better practice to explicitly request its closure with a call to `CloseA` (see page 614). This also allows the caller to specify the type of closure to perform. Releasing all references to the asynchronous socket will always perform a forced close.

Events related to this interface are reported through the `IAsyncSocketMgr` (see page 624) interface.

This interface MUST be obtained by calling `CAsyncSocketFactory::CreateAsyncSocket` (see page 524).

Location

Network/IAsyncSocket.h

See Also

`IAsyncClientSocket` (see page 599) `IAsyncIoSocket` (see page 602) `IAsyncServerSocket` (see page 607) `IAsyncSocketMgr` (see page 624) `IAsyncUnconnectedIoSocket` (see page 631) `CAsyncSocketFactory::CreateAsyncSocket` (see page 524)

Methods

Method	Description
• A <code>Activate</code> (see page 613)	Activates the socket on the provided ServicingThread.
• A <code>CloseA</code> (see page 614)	Closes the asynchronous socket.
• A <code>EraseAllUserInfo</code> (see page 614)	Erases all user information stored in the socket.
• A <code>EraseUserInfo</code> (see page 615)	Erases a specific user information stored in the socket.
• A <code>GetHandle</code> (see page 615)	Returns the socket handle.
• A <code>GetLocalAddress</code> (see page 615)	Retrieves the local address to which the socket is bound.
• A <code>GetLocalInterfaceAddress</code> (see page 616)	Retrieves the local address to which the socket is bound on a local interface.
• A <code>GetOpaque</code> (see page 616)	Retrieves the opaque identifier.
• A <code>GetServicingThreadIEcomUnknown</code> (see page 616)	Returns a pointer to the <code>IEComUnknown</code> (see page 416) interface of the servicing thread that has been used to activate the socket.
• A <code>GetSocketType</code> (see page 617)	Retrieves the type of the socket.
• A <code>GetUserInfo</code> (see page 618)	Retrieves a specific user information stored in the socket.
• A <code>InsertUserInfo</code> (see page 618)	Inserts a specific user information to be stored in the socket.
• A <code>SetAsyncSocketMgr</code> (see page 619)	Sets the manager associated with the interface <code>IAsyncSocket</code> .
• A <code>SetOpaque</code> (see page 619)	Sets the user opaque value.
• A <code>SetSocketType</code> (see page 619)	Sets the socket type from a string array.

IEComUnknown Class

IEComUnknown Class	Description
• A <code>AddRef</code> (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A <code>QueryIf</code> (see page 418)	Queries an object for a supported interface.
• A <code>ReleaseRef</code> (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.20.1 - Methods

2.11.1.20.1.1 - `IAsyncSocket::Activate` Method

Activates the socket on the provided ServicingThread.

C++

```
virtual mxt_result Activate(IN IEComUnknown* pIEComUnknown) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pIEComUnknown	A pointer to an interface serviced by a Servicing Thread. It can be NULL, in which case a new servicing thread will be created for the socket.

Returns

resS_OK: The socket activated correctly. resFE_INVALID_ARGUMENT: pIEComUnknown is NULL. resFE_INVALID_STATE: The socket has already been activated.

Description

This method must be the first method to be called on a socket. It is used to associate the socket with a servicing thread. You will have to call SetAsyncSocketMgr (see page 619) and SetOpaque (see page 619) before you try to use the socket but after you called this method.

When this interface is properly obtained by calling CAsyncSocketFactory::CreateAsyncSocket (see page 524), calling this method is unnecessary because the servicing thread is already activated.

See Also

SetAsyncSocketMgr (see page 619), SetOpaque (see page 619), CAsyncSocketFactory::CreateAsyncSocket (see page 524)

2.11.1.20.1.2 - IAsyncSocket::CloseA Method

Closes the asynchronous socket.

C++

```
virtual mxt_result CloseA(IN ISocket::ECloseBehavior eCloseBehavior) = 0;
```

Parameters

Parameters	Description
IN ISocket::ECloseBehavior eCloseBehavior	How the socket is to be closed.

Returns

Always returns resS_OK. Errors are reported through the IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) event.

Description

IMPORTANT: This is an asynchronous method!

This is used to asynchronously close the socket. Once the socket is closed, the IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625) event will be generated back to the socket manager once the asynchronous closure has completed.

IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625) is always reported when the closure was successful.

IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625) is not reported over a reliable transport if the closure has failed.

IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) is reported in case of a failure over a reliable transport. Over an unreliable transport, IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625) is always reported once the closure has completed (even if the socket closure has failed).

After IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625) is generated, it is possible for the user of the socket to re-use the asynchronous socket.

Closing the socket has the effect of resetting all the socket's options to their default values. Thus, if the socket is reused, it might be necessary to reconfigure its socket options.

See Also

IAsyncSocketMgr::EvAsyncSocketMgrClosed (see page 625), IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625)

2.11.1.20.1.3 - IAsyncSocket::EraseAllUserInfo Method

Erases all user information stored in the socket.

C++

```
virtual mxt_result EraseAllUserInfo() = 0;
```

Returns

- resS_OK: The user information has been correctly erased.

Description

This method erases all user information currently stored in the socket. After calling this method, all user information contained in the socket is no longer be available for retrieval using IAsyncSocket::GetUserInfo (see page 618).

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is

no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.4 - IAsyncSocket::EraseUserInfo Method

Erases a specific user information stored in the socket.

C++

```
virtual mxt_result EraseUserInfo(IN const char* pszUserInfo) = 0;
```

Parameters

Parameters	Description
IN const char* pszUserInfo	The identifier of the user information to remove from the socket. This parameter must not be NULL.

Returns

- resS_OK: The user information has been correctly erased.
- resFE_INVALID_ARGUMENT: pszUserInfo is NULL.

Description

This method erases the specified user information in the socket. If the call succeeds, the specified user information is no longer available for retrieval using IAsyncSocket::GetUserInfo (see page 618).

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.5 - IAsyncSocket::GetHandle Method

Returns the socket handle.

C++

```
virtual mxt_result GetHandle(OUT mxt_hSocket* phSocket) const = 0;
```

Parameters

Parameters	Description
OUT mxt_hSocket* phSocket	A pointer to a socket handle. It cannot be NULL.

Returns

resS_OK: Handle returned successfully. resFE_INVALID_ARGUMENT: phSocket is NULL. resFE_INVALID_STATE: The socket is not bound.

Description

This method gets the handle of the underlying socket.

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.6 - IAsyncSocket::GetLocalAddress Method

Retrieves the local address to which the socket is bound.

C++

```
virtual mxt_result GetLocalAddress(OUT CSocketAddr* pLocalAddress) const = 0;
```

Parameters

Parameters	Description
OUT CSocketAddr* pLocalAddress	OUT (see page 39) parameter that will contain the address and port information. It cannot be NULL.

Returns

resS_OK: The local address has been returned. resFE_INVALID_STATE: The socket is not bound. resFE_INVALID_ARGUMENT: pLocalAddress is NULL.

Description

This method is used to retrieve the address and port to which the socket has been bound. This may be a port located on a remote server (e.g. Allocate using a TURN server), this may also be a port located on a NAT public side (e.g. Discovery using a STUN Binding Request) or a port located on a local interface. The user must be aware that the returned address and port is not always located on a local interface and hence, it SHOULD NOT be used in a future call to `IAsyncClientSocket` (see page 599) or `IAsyncServerSocket::BindA` (see page 608). `GetLocalInterfaceAddress` (see page 616) should be called instead to retrieve an address and port located on a local interface.

It will return an error until the socket is bound to an interface with the `IAsyncClientSocket` (see page 599) or `IAsyncServerSocket::BindA` (see page 608) method.

Note that this method is synchronous. The `IAsyncSocket` (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncSocket` (see page 612) interface.

2.11.1.20.1.7 - `IAsyncSocket::GetLocalInterfaceAddress` Method

Retrieves the local address to which the socket is bound on a local interface.

C++

```
virtual mxt_result GetLocalInterfaceAddress(OUT CSocketAddr* pLocalInterfaceAddress) const = 0;
```

Parameters

Parameters	Description
<code>pLocalAddress</code>	OUT (see page 39) parameter that will contain the address and port information. It cannot be NULL.

Returns

`resS_OK`: The local interface address has been returned. `resFE_INVALID_STATE`: The socket is not bound. `resFE_INVALID_ARGUMENT`: `pLocalAddress` is NULL.

Description

This method is used to retrieve the address and port to which the socket has been bound on a local interface. The method is different than `GetLocalAddress` (see page 615) in that the returned local address and port always reside on a local interface.

The method will return an error until the socket is bound to an interface with the `IAsyncClientSocket` (see page 599) or `IAsyncServerSocket::BindA` (see page 608) method.

Note that this method is synchronous. The `IAsyncSocket` (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncSocket` (see page 612) interface.

2.11.1.20.1.8 - `IAsyncSocket::GetOpaque` Method

Retrieves the opaque identifier.

C++

```
virtual mxt_result GetOpaque(OUT mxt_opaque* popq) const = 0;
```

Parameters

Parameters	Description
<code>OUT mxt_opaque* popq</code>	OUT (see page 39) parameter that receives the opaque identifier. It cannot be NULL.

Returns

`resS_OK`: The opaque value has been returned. `resFE_INVALID_ARGUMENT`: `popq` is NULL.

Description

This method is used to retrieve the opaque identifier of the socket.

Note that this method is synchronous. The `IAsyncSocket` (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncSocket` (see page 612) interface.

2.11.1.20.1.9 - `IAsyncSocket::GetServicingThreadIEcomUnknown` Method

Returns a pointer to the `IEComUnknown` (see page 416) interface of the servicing thread that has been used to activate the socket.

C++

```
virtual mxt_result GetServicingThreadIEcomUnknown(OUT IEComUnknown** ppIEComUnknown) const = 0;
```

Parameters

Parameters	Description
OUT IEComUnknown** ppIEComUnknown	A pointer to receive the IEComUnknown (see page 416) interface pointer of the ServicingThread that has been used to activate the socket. Note that while the parameter CANNOT be NULL, the value returned in it CAN. This will happen if the Activate (see page 613) method of this interface has not been called yet. The caller MUST call ReleaseIfRef (see page 420) on the returned interface when he is done with it.

Returns

resS_OK: The interface has been successfully returned. resFE_INVALID_ARGUMENT: The provided parameter is NULL. resFE_FAIL: The socket has not been activated. The value of the parameter will be NULL in this case.

Description

Returns a pointer to the IEComUnknown (see page 416) interface of the servicing thread that has been used to activate the socket.

See Also

Activate (see page 613)

2.11.1.20.1.10 - GetSocketType**2.11.1.20.1.10.1 - IAsyncSocket::GetSocketType Method**

Retrieves the type of the socket.

C++

```
virtual mxt_result GetSocketType(OUT ISocket::ESocketType* peSocketType) const = 0;
```

Parameters

Parameters	Description
OUT ISocket::ESocketType* peSocketType	OUT (see page 39) parameter that receives the socket type information, as defined in the ISocket::ESocketType (see page 652) enumeration. It cannot be NULL.

Returns

resS_OK: The socket type has been returned. resFE_INVALID_STATE: The socket is not bound. resFE_INVALID_ARGUMENT: peSocketType is NULL.

Description

This method is used to retrieve the connection type of the socket.

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.10.2 - IAsyncSocket::GetSocketType Method

Returns the socket type as a string array.

C++

```
virtual const char* const* GetSocketType(OUT unsigned int* puSize) const = 0;
```

Parameters

Parameters	Description
OUT unsigned int* puSize	The size of the returned string array. It cannot be NULL.

Returns

An array of strings describing the socket type. The return value can be NULL, if the socket type is not set. In this case, *puSize will be zero. The return value will be NULL, if puSize is NULL.

Description

This method returns the socket type as an array of strings. The array is in fact the same as the one given to the socket factory, except

for accepted sockets on which it will be derived from the server socket type.

Notes

If the application needs to keep a reference on the returned socket type array, it must take the precaution to copy the array's content (the `const char*` values and the strings themselves). This is because the socket's internal array might get reallocated at some point if the socket type is changed.

This method is synchronous. The `IAsyncSocket` (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncSocket` (see page 612) interface.

Example

- `{ "TCP", m=accepted" }; *puSize = 1`
- `{ "TLS" (see page 794), m=client", "TCP, m=client" }; *puSize =2`
- `{ "SIP", "TLS" (see page 794), m=server", "TCP, m=server" }; *puSize = 3`

2.11.1.20.1.11 - `IAsyncSocket::GetUserInfo` Method

Retrieves a specific user information stored in the socket.

C++

```
virtual mxt_result GetUserInfo(IN const char* pszUserInfo, OUT CBlob* pblob) const = 0;
```

Parameters

Parameters	Description
IN const char* pszUserInfo	The identifier of the user information to retrieve from the socket. This parameter must not be NULL.
OUT CBlob* pblob	A blob to receive the user information. This parameter must not be NULL.

Returns

- `resS_OK`: The user information has been correctly retrieved.
- `resFE_INVALID_ARGUMENT`: `pszUserInfo` or `pblob` is NULL.
- `resFE_FAIL`: The requested user information has not been found.

Description

This method retrieves the specified user information in the socket and copies it to the provided blob.

Note that this method is synchronous. The `IAsyncSocket` (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the `IAsyncSocket` (see page 612) interface.

2.11.1.20.1.12 - `IAsyncSocket::InsertUserInfo` Method

Inserts a specific user information to be stored in the socket.

C++

```
virtual mxt_result InsertUserInfo(IN const char* pszUserInfo, IN const CBlob* pblob) = 0;
```

Parameters

Parameters	Description
IN const char* pszUserInfo	The identifier of the user information to insert in the socket. This parameter must not be NULL.
IN const CBlob* pblob	The user information content to insert in the socket. This parameter must not be NULL.

Returns

- `resS_OK`: The user information has been correctly inserted.
- `resSI_FW_NET_REPLACED_USER_INFO_ENTRY`: The user information that already existed in the socket has been replaced.
- `resFE_INVALID_ARGUMENT`: `pszUserInfo` or `pblob` is NULL.
- `resFE_FAIL`: The user information could not be inserted.

Description

This method inserts the specified user information in the socket by copying the provided blob to an internal one that is associated with

the provided identifier. If there is already an entry with the same identifier present in the socket, the existing data associated with it is replaced by the new provided data and the method returns resSI_FW_NET_REPLACE_USER_INFO_ENTRY. If a new entry is effectively inserted, resS_OK is returned.

The user information mechanism is meant for applications to be able to identify the sockets when they are retrieved through CAAsyncSocketFactory::GetSocketList (see page 524).

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.13 - IAsyncSocket::SetAsyncSocketMgr Method

Sets the manager associated with the interface IAsyncSocket (see page 612).

C++

```
virtual mxt_result SetAsyncSocketMgr(IN IAsyncSocketMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncSocketMgr* pMgr	Pointer to a IAsyncSocketMgr (see page 624) manager. It cannot be NULL.

Returns

resS_OK: The manager has been set correctly. resFE_INVALID_STATE: The manager has probably already been set.
resFE_INVALID_ARGUMENT: pMgr is NULL.

Description

This method sets the manager associated with the IAsyncSocket (see page 612) interface.

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

See Also

IAsyncSocketMgr (see page 624)

2.11.1.20.1.14 - IAsyncSocket::SetOpaque Method

Sets the user opaque value.

C++

```
virtual mxt_result SetOpaque(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	The identifier that must be associated with the socket.

Returns

Always returns resS_OK.

Description

This method sets the opaque value that must be associated with the socket. This opaque value is sent with each event to the managers. A call to this method will always replace the existing value. This value only has meaning for the application.

Note that this method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.20.1.15 - IAsyncSocket::SetSocketType Method

Sets the socket type from a string array.

C++

```
virtual mxt_result SetSocketType(IN const char* const* ppszType, IN unsigned int uTypeSize) = 0;
```

Parameters

Parameters	Description
IN const char* const* ppszType	The array of strings describing the socket type. The syntax of this parameter is the same as that of the apszType parameter to the CAsyncSocketFactory::CreateAsyncSocket (see page 524) method. It cannot be NULL.
IN unsigned int uTypeSize	The number of elements in the string arrays passed in the ppszType parameter. This parameter must not be 0.

Returns

- resS_OK: The socket type has correctly been set.
- resFE_INVALID_ARGUMENT: ppszType is NULL or uTypeSize is 0.

Description

This method sets the socket type of the socket. This method is normally only used by the CAsyncSocketFactory (see page 523) or the server sockets. It is called as the last step of the asynchronous socket creation process, that is, after all creation managers have been called and one of them returned a valid socket.

Notes

Internally the socket allocates a new array and copies the string content. This internal array will be deleted and reallocated.

This method is synchronous. The IAsyncSocket (see page 612) will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncSocket (see page 612) interface.

2.11.1.21 - IAsyncSocketBufferSizeOptions Class

Interface defining the socket options that are related to buffer size.

Class Hierarchy



C++

```
class IASyncSocketBufferSizeOptions : public IECOMUnknown;
```

Description

This interface defines the socket options that are related to buffer size.

Location

Network/IAsyncSocketBufferSizeOptions.h

See Also

IAsyncSocketUdpOptions (see page 629) IAsyncSocketTcpOptions (see page 627) IAsyncSocketQualityOfServerOptions
IAsyncSocketWindowsGqosOptions (see page 630)

Methods

Method	Description
• A SetReceiveBufferSize (see page 621)	Sets a socket level option which determines the maximum socket receive buffer size.
• A SetTransmitBufferSize (see page 621)	Sets a socket level option which determines the maximum socket transmit buffer size.

IECOMUnknown Class

IECOMUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
A	abstract

2.11.1.21.1 - Methods

2.11.1.21.1.1 - IAsyncSocketBufferSizeOptions::SetReceiveBufferSize Method

Sets a socket level option which determines the maximum socket receive buffer size.

C++

```
virtual mxt_result SetReceiveBufferSize(IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSize	The size of the receive buffer.

Returns

resS_OK: The receive buffer size has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines the maximum socket receive buffer size in bytes. Usually the default value of this option is adequate; only experts should change this value. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.21.1.2 - IAsyncSocketBufferSizeOptions::SetTransmitBufferSize Method

Sets a socket level option which determines the maximum socket transmit buffer size.

C++

```
virtual mxt_result SetTransmitBufferSize(IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSize	The size of the transmit buffer.

Returns

resS_OK: The transmit buffer size has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines the maximum socket transmit buffer size in bytes. Usually the default value of this option is adequate; only experts should change this value. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.22 - IAsyncSocketFactoryConfigurationMgr Class

This interface is used to receive asynchronous sockets configuration requests from the socket factory.

Class Hierarchy

```
[IAsyncSocketFactoryConfigurationMgr]
```

C++

```
class IAsyncSocketFactoryConfigurationMgr;
```

Description

This interface is a manager which handles the configuration of asynchronous sockets. The configuration manager is notified when an asynchronous socket has just been created and must be configured with various socket options. In contrast with the creation managers, all the configuration managers are notified and have the opportunity to configure the asynchronous socket. The configuration managers are called in their registration ordering so that the last registered configuration manager may override configurations previously performed by other managers. To be called, the manager must be registered using CAsyncSocketFactory::RegisterConfigurationMgr (see page 525).

Location

Network/IAsyncSocketFactoryConfigurationMgr.h

See Also

CAsyncSocketFactory (see page 523), IAsyncSocketFactoryCreationMgr (see page 622)

Methods

Method	Description
EvConfigurationRequested (see page 622)	Notifies the manager that a newly created asynchronous socket must be configured.

Legend

	Method
	abstract

2.11.1.22.1 - Methods**2.11.1.22.1.1 - IAsyncSocketFactoryConfigurationMgr::EvConfigurationRequested Method**

Notifies the manager that a newly created asynchronous socket must be configured.

C++

```
virtual mxt_result EvConfigurationRequested(IN const char* const* apszType, IN unsigned int uTypeSize, INOUT IAsyncSocket* pAsyncSocket) = 0;
```

Parameters

Parameters	Description
IN const char* const* apszType	An array of strings representing the type of the socket.
IN unsigned int uTypeSize	The size of the array of strings.
INOUT IAsyncSocket* pAsyncSocket	The newly created asynchronous socket that must be configured.

Returns

resSI_FALSE: The asynchronous socket has not been configured. resSI_TRUE: The asynchronous socket has been configured.
resFE_FAIL: An error has been detected while configuring the asynchronous socket.

Description

This method notifies the manager that a newly created asynchronous socket must be configured. The type of the asynchronous socket that has been created is provided as a sequence of network protocols starting from the highest level down to the lowest level. Each level is possibly followed by arguments.

Example

```
{
    "RTCP", "UDP" }, 2
{
    "RTP", "UDP" }, 2
{
    "RTP_RTCP", "UDP" }, 2
{
    "SIP", "TCP, m=client" }, 2
{
    "SIP", "TCP, m=server" }, 2
{
    "SIP", "TLS, m=client", "TCP, m=client" }, 3
{
    "SIP", "TLS, m=server", "TCP, m=server" }, 3
{
    "SIP", "UDP" }, 2
{
    "STUN", "TCP, m=client" }, 2
{
    "STUN", "TCP, m=server" }, 2
{
    "STUN", "TLS, m=client", "TCP" }, 3
{
    "STUN", "TLS, m=server", "TCP" }, 3
{
    "STUN", "UDP" }, 2
}
```

2.11.1.23 - IAsyncSocketFactoryCreationMgr Class

This interface is used to receive asynchronous sockets creation requests from the socket factory.

Class Hierarchy

IAsyncSocketFactoryCreationMgr

C++

```
class IAsyncSocketFactoryCreationMgr;
```

Description

This interface is a manager which handles the creation of asynchronous sockets. The manager is called each time a new asynchronous socket needs to be created. To be called, the manager must be registered using CAsyncSocketFactory::RegisterCreationMgr (see page 525). Created sockets must be ECOM (see page 412) objects that implement the following required interfaces and possibly one

or more optional interfaces.

Required interfaces for all sockets:

- IAsyncSocket (see page 612).

Required interfaces based on the socket's usage:

- Client sockets:
 - IAsyncClientSocket (see page 599)
 - IAsynccloSocket (see page 602)
- Client sockets implementing the TLS (see page 794) protocol:
 - All client sockets interfaces plus:
 - IAsyncTlsSocket (see page 812)
- Server sockets:
 - IAsyncServerSocket (see page 607)
- Server sockets implementing the TLS (see page 794) protocol:
 - All server sockets interfaces plus:
 - IAsyncTlsServerSocket (see page 811)

Optional interfaces:

- IAsyncSocketUdpOptions (see page 629)
- IAsyncSocketTcpOptions (see page 627)
- IAsyncSocketBufferSizeOptions (see page 620)
- IAsyncSocketQualityOfServiceOptions (see page 625)
- IAsyncSocketWindowsGqosOptions (see page 630)

Optional interfaces for sockets implementing the TLS protocol:

- IAsyncSocketTlsRenegotiation

Location

Network/IAsyncSocketFactoryCreationMgr.h

See Also

CAsyncSocketFactory (see page 523), IAsyncSocketFactoryConfigurationMgr (see page 621)

Methods

Method	Description
EvCreationRequested (see page 623)	Notifies the manager that a new asynchronous socket must be created.

Legend

	Method
	abstract

2.11.1.23.1 - Methods

2.11.1.23.1.1 - IAsyncSocketFactoryCreationMgr::EvCreationRequested Method

Notifies the manager that a new asynchronous socket must be created.

C++

```
virtual mxt_result EvCreationRequested(IN IEComUnknown* pServicingThread, IN const char* const* apszType, IN
unsigned int uTypeSize, OUT IAsyncSocket** ppAsyncSocket) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pServicingThread	The servicing thread used to activate the asynchronous socket. If NULL is passed, the asynchronous socket automatically creates its own servicing thread with default parameters.
IN const char* const* apszType	An array of strings representing the type of the socket.
IN unsigned int uTypeSize	The size of the array of string.
OUT IAsyncSocket** ppAsyncSocket	The location where the new asynchronous socket is returned.

Returns

resSI_FALSE: The asynchronous socket has not been created. resSI_TRUE: The asynchronous socket has been created. resFE_FAIL: An error has been detected while creating the asynchronous socket.

Description

This method notifies the manager that a new asynchronous socket must be created. Created sockets must be ECOM (see page 412) objects. The type of the asynchronous socket to be created is provided as a sequence of network protocols starting from the highest level down to the lowest level. Each level is possibly followed by arguments.

Example

```
{
  "RTCP", "UDP" }, 2
{
  "RTP", "UDP" }, 2
{
  "RTCP_RTCP", "UDP" }, 2
{
  "SIP", "TCP, m=client" }, 2
{
  "SIP", "TCP, m=server" }, 2
{
  "SIP", "TLS, m=client", "TCP, m=client" }, 3
{
  "SIP", "TLS, m=server", "TCP, m=server" }, 3
{
  "SIP", "UDP" }, 2
{
  "STUN", "TCP, m=client" }, 2
{
  "STUN", "TCP, m=server" }, 2
{
  "STUN", "TLS, m=client", "TCP" }, 3
{
  "STUN", "TLS, m=server", "TCP" }, 3
{
  "STUN", "UDP" }, 2
```

2.11.1.24 - IAsyncSocketMgr Class

Defines basic events that can be reported by all types of asynchronous sockets.

Class Hierarchy

```
IAsyncSocketMgr
```

C++

```
class IAsyncSocketMgr;
```

Description

This is the basic asynchronous socket manager interface that must be inherited by all users of asynchronous sockets.

This interface defines basic events that can be reported by all types of asynchronous sockets.

Location

Network/IAsyncSocketMgr.h

See Also

IAsyncClientSocket (see page 599) IAsyncloSocket (see page 602) IAsyncServerSocket (see page 607) IAsyncSocket (see page 612) IAsyncUnconnectedloSocket (see page 631)

Methods

Method	Description
◆ A EvAsyncSocketMgrClosed (see page 625)	Notifies of the closure of the socket.
◆ A EvAsyncSocketMgrClosedByPeer (see page 625)	Notifies of the closure of the socket by the peer.
◆ A EvAsyncSocketMgrErrorDetected (see page 625)	Reports an error detected on the socket.

Legend

	Method
	abstract

2.11.1.24.1 - Methods**2.11.1.24.1.1 - IAsyncSocketMgr::EvAsyncSocketMgrClosed Method**

Notifies of the closure of the socket.

C++

```
virtual void EvAsyncSocketMgrClosed(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket that has been closed.

Description

This event is reported by the socket after it has been locally closed.

See Also

IAsyncSocket::CloseA (see page 614)

2.11.1.24.1.2 - IAsyncSocketMgr::EvAsyncSocketMgrClosedByPeer Method

Notifies of the closure of the socket by the peer.

C++

```
virtual void EvAsyncSocketMgrClosedByPeer(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with this socket that has been closed.

Description

This event is reported by the socket after it has been closed by the peer to which it was connected.

Note that this can be reported for sockets of type eSTREAM and eSEQPACKET.

See Also

IAsyncSocket::CloseA (see page 614)

2.11.1.24.1.3 - IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected Method

Reports an error detected on the socket.

C++

```
virtual void EvAsyncSocketMgrErrorDetected(IN mxt_opaque opq, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket in error.
IN mxt_result res	The error detected on the socket.

Description

This event can be generated by a socket when it detects an error condition for an asynchronous call, like CloseA, ConnectA, AcceptA.

2.11.1.25 - IAsyncSocketQualityOfServiceOptions Class

Interface defining the socket options that are related to quality of service.

Class Hierarchy



C++

```
class IAsyncSocketQualityOfServiceOptions : public IEComUnknown;
```

Description

This interface defines the socket options that are related to quality of service.

Location

Network/IAsyncSocketBufferSizeOptions.h

See Also

IAsyncSocketUdpOptions (see page 629) IAsyncSocketTcpOptions (see page 627) IAsyncSocketBufferSizeOptions (see page 620)
IAsyncSocketWindowsGqosOptions (see page 630)

Methods

Method	Description
• A Set8021QUserPriority (see page 626)	Sets a socket level option which enables/disables utilization of the 802.1Q user priority.
• A SetTos (see page 627)	Sets a socket level option which sets the TOS (see page 39) byte field in the IP protocol header.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.25.1 - Methods

2.11.1.25.1.1 - IAsyncSocketQualityOfServiceOptions::Set8021QUserPriority Method

Sets a socket level option which enables/disables utilization of the 802.1Q user priority.

C++

```
virtual mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	Determines whether the option is enabled or not.
IN uint8_t uUserPriority	The value to use in the 802.1Q user priority field.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which enables/disables utilization of the 802.1Q user priority. If the option is enabled, then the 802.1Q user priority field is set to uUserPriority. This method can be called even if the socket is not bound yet.

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Nucleus), the execution of this method will output an error trace before returning with resS_OK.

2.11.1.25.1.2 - IAsyncSocketQualityOfServiceOptions::SetTos Method

Sets a socket level option which sets the TOS (see page 39) byte field in the IP protocol header.

C++

```
virtual mxt_result SetTos(IN uint8_t uTos) = 0;
```

Parameters

Parameters	Description
IN uint8_t uTos	The value of the TOS (see page 39) byte field for this socket.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which sets the TOS (see page 39) byte field in the IP protocol header. This method can be called even if the socket is not bound yet.

Windows platforms specific information:

For this option to work under Windows 2000, XP, and 2003 Server, the value **DisableUserTOSSetting** must be set to 0 in the registry.

It is located in the following key: HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters

The value is a REG_DWORD that must be set to 0 in order for the TOS (see page 39) flags to be set in the IP header. On Windows 2000 and some older builds of Windows XP, the value must be added. Windows must be rebooted for the changes to take effect.

Additional information can be found here: <http://support.microsoft.com/default.aspx?scid=kb;en-us;248611>

Notes: This option is target dependent. If it is compiled for a target that does not support the option (like Windows CE), the execution of this method will output an error trace before returning with resS_OK.

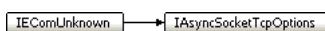
In Windows CE 4.0, the function call succeeds but the TOS (see page 39) byte field in the IP protocol header will remain unchanged.

Under Nucleus, this method sets the DSCP bytefield instead of the TOS (see page 39). It is impossible to set the TOS (see page 39) byte field.

2.11.1.26 - IAsyncSocketTcpOptions Class

Interface defining the options that are configurable on a TCP socket.

Class Hierarchy



C++

```
class IAsyncSocketTcpOptions : public IEComUnknown;
```

Description

This interface defines the options that are configurable on a TCP socket. There are two configurable options: the keepalive and the Nagle algorithm options. The reuse address option is automatically set and is not configurable.

Location

Network/IAsyncSocketTcpOptions.h

See Also

IAsyncSocketUdpOptions (see page 629) IAsyncSocketBufferSizeOptions (see page 620) IAsyncSocketQualityOfServerOptions
IAsyncSocketWindowsGqosOptions (see page 630)

Methods

Method	Description
SetConnectTimeoutMs (see page 628)	Sets the connect timeout in ms.
SetKeepAlive (see page 628)	Enables/disables the TCP keep-alive option.
SetNagle (see page 628)	Enables/disables the TCP Nagle algorithm option.

IEComUnknown Class

IEComUnknown Class	Description
• A AddIfRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseIfRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.11.1.26.1 - Methods**2.11.1.26.1.1 - IAsyncSocketTcpOptions::SetConnectTimeoutMs Method**

Sets the connect timeout in ms.

C++

```
virtual void SetConnectTimeoutMs(IN uint64_t uConnectTimeoutMs) = 0;
```

Parameters

Parameters	Description
IN uint64_t uConnectTimeoutMs	The connect timeout.

Description

Sets the connect timeout in ms. If a TCP socket is not connected when the timeout occurs, an error is reported to the manager. If the OS timer is smaller than this value, it has priority.

2.11.1.26.1.2 - IAsyncSocketTcpOptions::SetKeepAlive Method

Enables/disables the TCP keep-alive option.

C++

```
virtual mxt_result SetKeepAlive(IN bool bEnable) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	If true, enables the TCP keep-alive messages on the socket.

Returns

resS_OK: The option has been successfully enabled. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to enable/disable the TCP socket option to send keep-alive messages on an idle TCP connection, using the operating system's default settings.

2.11.1.26.1.3 - IAsyncSocketTcpOptions::SetNagle Method

Enables/disables the TCP Nagle algorithm option.

C++

```
virtual mxt_result SetNagle(IN bool bEnable) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	If true, enables the TCP Nagle algorithm on the socket.

Returns

resS_OK: The option has been successfully enabled. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to enable/disable the TCP Nagle algorithm option. The Nagle algorithm controls how the TCP stream is packed into TCP fragment.

2.11.1.27 - IAsyncSocketUdpOptions Class

Interface defining the options that are configurable on a UDP socket.

Class Hierarchy



C++

```
class IAsyncSocketUdpOptions : public IEComUnknown;
```

Description

This interface defines the options that are configurable on a UDP socket. There is one configurable option: the broadcast option. The reuse address option is automatically set and is not configurable.

Location

Network/IAsyncSocketUdpOptions.h

See Also

IAsyncSocketTcpOptions (see page 627) IAsyncSocketBufferSizeOptions (see page 620) IAsyncSocketQualityOfServerOptions
IAsyncSocketWindowsGqosOptions (see page 630)

Methods

Method	Description
• A SetBroadcast (see page 629)	Enables/disables the UDP broadcast option.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.27.1 - Methods

2.11.1.27.1.1 - IAsyncSocketUdpOptions::SetBroadcast Method

Enables/disables the UDP broadcast option.

C++

```
virtual mxt_result SetBroadcast(IN bool bEnable) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	If true, enable the UDP broadcast messages on the socket.

Returns

resS_OK: The option has been successfully enabled. See GetSocketErrorId (see page 656) for possible return values.

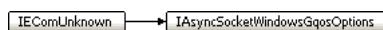
Description

This method is used to enable/disable the UDP socket option to broadcast messages on the socket. This method can be called even if the socket is not bound yet.

2.11.1.28 - IAsyncSocketWindowsGqosOptions Class

Interface defining the socket options that are related to quality of service. This interface is specialized for the Windows GQOS flowspec option.

Class Hierarchy



C++

```
class IAsyncSocketWindowsGqosOptions : public IEComUnknown;
```

Description

This interface defines the socket options that are related to quality of service. This interface is specialized for the Windows GQOS flowspec option.

Location

Network/IAsyncSocketWindowsGqosOptions.h

See Also

IAsyncSocketUdpOptions (see page 629) IAsyncSocketTcpOptions (see page 627) IAsyncSocketBufferSizeOptions (see page 620)
IAsyncSocketQualityOfServerOptions

Methods

Method	Description
• A SetWindowsReceivingFlowspec (see page 630)	Sets the receiving flowspec (Windows only).
• A SetWindowsSendingFlowspec (see page 630)	Sets the sending flowspec (Windows only).

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.28.1 - Methods

2.11.1.28.1.1 - IAsyncSocketWindowsGqosOptions::SetWindowsReceivingFlowspec Method

Sets the receiving flowspec (Windows only).

C++

```
virtual mxt_result SetWindowsReceivingFlowspec(IN FLOWSPEC* pReceivingFlowspec) = 0;
```

Parameters

Parameters	Description
IN FLOWSPEC* pReceivingFlowspec	The value of the receiving flowspec.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets the receiving flowspec (Windows only). This method can be called even if the socket is not bound yet.

2.11.1.28.1.2 - IAsyncSocketWindowsGqosOptions::SetWindowsSendingFlowspec Method

Sets the sending flowspec (Windows only).

C++

```
virtual mxt_result SetWindowsSendingFlowspec(IN FLOWSPEC* pSendingFlowspec) = 0;
```

Parameters

Parameters	Description
IN FLOWSPEC* pSendingFlowspec	The value of the sending flow spec.

Returns

resS_OK: The option has been successfully set. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets the sending flowspec (Windows only). This method can be called even if the socket is not bound yet.

2.11.1.29 - IAsyncUnconnectedIoSocket Class

Interface defining the methods accessible on unconnected asynchronous sockets to send and receive data.

Class Hierarchy**C++**

```
class IAsyncUnconnectedIoSocket : public IEComUnknown;
```

Description

The IAsyncUnconnectedIoSocket interface defines the methods accessible on an unconnected asynchronous socket that can be used to send and receive data.

Events related to this interface are reported through the IAsyncUnconnectedIoSocketMgr (see page 634) interface:

- The IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToRecv (see page 635) event is generated when there is data ready for reception on the socket. This means that calling IAsyncUnconnectedIoSocket::RecvFrom (see page 632) should yield incoming data. IAsyncUnconnectedIoSocket::RecvFrom (see page 632) must be called repeatedly until it returns with zero bytes received. If this condition is not reached, the event will never be reported again.
- The IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToSend (see page 635) event is generated once the socket becomes writable after a call to IAsyncUnconnectedIoSocket::SendTo (see page 633) has returned successfully but with less than the provided buffer size bytes sent.

Although the interface name seems to specify that the IAsyncUnconnectedIoSocket::RecvFrom (see page 632) and IAsyncUnconnectedIoSocket::SendTo (see page 633) methods are asynchronous, this is not the case. These methods are synchronous because the socket should not queue data. Queueing data in the socket could end up with a non deterministic message queue allocation. The asynchronous part is managed through the asynchronous notification of the IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToRecv (see page 635) and IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToSend (see page 635). It is guaranteed that these two events will never be reported to the manager while a send or receive operation is in progress.

Location

Network/IAsyncUnconnectedIoSocket.h

See Also

IAsyncClientSocket (see page 599) IAsyncIoSocket (see page 602) IAsyncServerSocket (see page 607) IAsyncSocket (see page 612) IAsyncUnconnectedIoSocketMgr (see page 634)

Methods

Method	Description
► A RecvFrom (see page 632)	Receives data from an unconnected socket.
► A SendTo (see page 633)	Sends data on an unconnected socket.
► A SetAsyncUnconnectedIoSocketMgr (see page 634)	Sets the manager associated with the interface IAsyncUnconnectedIoSocket.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
A	abstract

2.11.1.29.1 - Methods

2.11.1.29.1.1 - RecvFrom

2.11.1.29.1.1.1 - IAsyncUnconnectedIoSocket::RecvFrom Method

Receives data from an unconnected socket.

C++

```
virtual mxt_result RecvFrom(OUT CBlob* pData, OUT CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pData	Blob object that will be filled with the data received on the socket. It cannot be NULL.
OUT CSocketAddr* pPeerAddress	Address from which the data has been received. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is either not bound or connected. resFE_INVALID_ARGUMENT: pData or pPeerAddress is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to receive data from an unconnected socket. Although this method is usually called when the socket user has been notified that there is data available on the socket, it can also be called without having received such a notification. The IAsyncUnconnectedIoSocket (see page 631) implementation notifies its user of the presence of incoming data through the IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketReadyToRecv (see page 635) event.

The maximum number of bytes that can be received is defined by the blob's capacity. The user of the socket might want to resize the blob to a specific size before calling IAsyncUnconnectedIoSocket::RecvFrom, as its capacity will not be changed by this call.

Note that this method is synchronous. The IAsyncUnconnectedIoSocket (see page 631) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncUnconnectedIoSocket (see page 631) interface.

See Also

IAsyncIoSocket::Recv (see page 603)

2.11.1.29.1.1.2 - IAsyncUnconnectedIoSocket::RecvFrom Method

Receives data from an unconnected socket.

C++

```
virtual mxt_result RecvFrom(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize, OUT CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puData	Pointer to a buffer where the received data is to be stored. It cannot be NULL.
IN unsigned int uCapacity	The maximum size available in the buffer pointed to by puData.
OUT unsigned int* puSize	OUT (see page 39) parameter that contains the number of bytes actually written in the buffer. It cannot be NULL.

OUT CSocketAddr* pPeerAddress

Address from which the data has been received. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is either not bound or connected. resFE_INVALID_ARGUMENT: pData, puSize or pPeerAddress is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to receive data from an unconnected socket. Although this method is usually called when the socket user has been notified that there is data available on the socket, it can also be called without having received such a notification. The IAsyncUnconnectedIoSocket (see page 631) implementation notifies its user of the presence of incoming data through the IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToRecv (see page 635) event.

Note that this method is synchronous. The IAsyncUnconnectedIoSocket (see page 631) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncUnconnectedIoSocket (see page 631) interface.

See Also

IAsyncIoSocket::Recv (see page 603)

2.11.1.29.1.2 - SendTo**2.11.1.29.1.2.1 - IAsyncUnconnectedIoSocket::SendTo Method**

Sends data on an unconnected socket.

C++

```
virtual mxt_result SendTo(IN const CBlob* pData, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Pointer to a blob containing the data to be sent. It cannot be NULL.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. It cannot be NULL. The value returned in this parameter is not reliable when the method returns anything other than resS_OK.
IN const CSocketAddr* pPeerAddress	Address to which the data must be sent. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is either not bound or connected. resFE_INVALID_ARGUMENT: pData, puSizeSent or pPeerAddress is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to send data on an unconnected socket.

Note that this method is synchronous. The IAsyncUnconnectedIoSocket (see page 631) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncUnconnectedIoSocket (see page 631) interface.

An application MUST not rely on the value of puSizeSent when this method returns anything other than resS_OK.

See Also

IAsyncIoSocket::Send (see page 604)

2.11.1.29.1.2.2 - IAsyncUnconnectedIoSocket::SendTo Method

Sends data on an unconnected socket.

C++

```
virtual mxt_result SendTo(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to a buffer containing the data to send. It cannot be NULL.

IN unsigned int uSize	The number of bytes in the buffer that must be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. It cannot be NULL. The value returned in this parameter is not reliable when the method returns anything other than resS_OK.
IN const CSocketAddr* pPeerAddress	Address to which the data must be sent. It cannot be NULL.

Returns

resFE_INVALID_STATE: The socket is either not bound or connected. resFE_INVALID_ARGUMENT: puData, puSizeSent or pPeerAddress is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to send data on an unconnected socket.

Note that this method is synchronous. The IAsyncUnconnectedIoSocket (see page 631) implementation will synchronize the various threads accessing it, thus there is no need to externally synchronize the access to the IAsyncUnconnectedIoSocket (see page 631) interface.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

IAsyncIoSocket::Send (see page 604)

2.11.1.29.1.3 - IAsyncUnconnectedIoSocket::SetAsyncUnconnectedIoSocketMgr Method

Sets the manager associated with the interface IAsyncUnconnectedIoSocket (see page 631).

C++

```
virtual mxt_result SetAsyncUnconnectedIoSocketMgr(IN IAsyncUnconnectedIoSocketMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncUnconnectedIoSocketMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

resFE_INVALID_STATE: The manager has probably already been set. resFE_INVALID_ARGUMENT: pMgr is NULL. See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets the manager associated with the interface IAsyncUnconnectedIoSocket (see page 631).

2.11.1.30 - IAsyncUnconnectedIoSocketMgr Class

This is the interface through which the unconnected asynchronous socket implementations report their events.

Class Hierarchy

IAsyncUnconnectedIoSocketMgr

C++

```
class IAsyncUnconnectedIoSocketMgr;
```

Description

This is the interface through which the unconnected asynchronous socket implementations report their events. All events are reported asynchronously with respect to the manager's execution context.

Location

Network/IAsyncUnconnectedIoSocketMgr.h

See Also

IAsyncUnconnectedIoSocket (see page 631)

Methods

Method	Description
EvAsyncUnconnectedIoSocketMgrReadyToRecv (see page 635)	Notifies the socket user that there is data available to be received on the socket.
EvAsyncUnconnectedIoSocketMgrReadyToSend (see page 635)	Notifies the socket user that the socket is ready to send data on the network.

Legend

	Method
	abstract

2.11.1.30.1 - Methods**2.11.1.30.1.1 -****IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToRecv Method**

Notifies the socket user that there is data available to be received on the socket.

C++

```
virtual void EvAsyncUnconnectedIoSocketMgrReadyToRecv(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque parameter associated with the socket.

Description

This event is reported by a socket when there is data available on the socket for reading.

Note that there might be some implementations of IAsyncUnconnectedIoSocket (see page 631) that do generate this event with a call to IAsyncUnconnectedIoSocket::RecvFrom (see page 632) yielding zero bytes received. Thus, the socket user should be ready to handle this case.

See Also

IAsyncIoSocket::Recv (see page 603) IAsyncUnconnectedIoSocket::RecvFrom (see page 632)

2.11.1.30.1.2 -**IAsyncUnconnectedIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToSend Method**

Notifies the socket user that the socket is ready to send data on the network.

C++

```
virtual void EvAsyncUnconnectedIoSocketMgrReadyToSend(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque parameter associated with the socket.

Description

This event is reported by a socket when it is possible for its user to send data on the network. Thus, after receiving this, the socket user should be able to call IAsyncUnconnectedIoSocket::SendTo (see page 633).

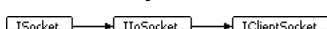
Note that there might be some implementations of IAsyncUnconnectedIoSocket (see page 631) that do generate this event even if a call to IAsyncUnconnectedIoSocket::SendTo (see page 633) returns with non zero bytes sent. The socket user must be able to handle this case.

See Also

IAsyncIoSocket::Send (see page 604) IAsyncUnconnectedIoSocket::SendTo (see page 633)

2.11.1.31 - IClientSocket Class

Interface defining the methods associated with a client socket.

Class Hierarchy**C++**

```
class IClientSocket : public IIoSocket;
```

Description

The IClientSocket interface defines the methods associated with a client socket. A client socket can be used to send and receive data, as defined in the IloSocket (see page 637) interface, and it can also connect to a remote address.

This interface only defines the Bind (see page 636) and Connect (see page 637) methods. Bind (see page 636) is used to bind the socket to a local address. Connect (see page 637) is used to connect a socket to a remote address. Note that some types of sockets must be connected in order to send and receive data (sockets of type eSTREAM and eSEQPACKET) while others can be used unconnected (sockets of type eDATAGRAM).

Location

Network/IClientSocket.h

See Also

ISocket (see page 646), IServerSocket (see page 644), IloSocket (see page 637)

Methods

Method	Description
• A Bind (see page 636)	Binds the socket to a local address.
• A Connect (see page 637)	Connects the socket to a remote address.

IloSocket Class

IloSocket Class	Description
• A GetPeerAddress (see page 638)	Retrieves the remote address to which the socket is connected.
• A Recv (see page 638)	Receives data from a connected socket.
• A RecvFrom (see page 639)	Receives data from a non-connected socket.
• A Send (see page 640)	Sends data on a connected socket.
• A SendTo (see page 641)	Sends data on a non-connected socket.

ISocket Class

ISocket Class	Description
• A Close (see page 647)	Closes a socket.
• A GetAddressFamily (see page 647)	Retrieves the address family of this socket.
• A GetLocalAddress (see page 647)	Retrieves the local address to which the socket is bound.
• A GetProtocolFamily (see page 648)	Retrieves the protocol family of this socket.
• A GetSocketType (see page 648)	Retrieves information about the socket and its transport.
• A Release (see page 648)	Deletes this socket.
• A Set8021QUserPriority (see page 649)	Configures the 802.1Q user priority for this socket.
• A SetTos (see page 649)	Configures the type of service (TOS (see page 39) associated with this socket.

Legend

•	Method
A	abstract

2.11.1.31.1 - Methods

2.11.1.31.1.1 - IClientSocket::Bind Method

Binds the socket to a local address.

C++

```
virtual mxt_result Bind(IN const CSocketAddr* pLocalAddress, OUT CSocketAddr* pEffectiveLocalAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the the local port. If the parameter is not NULL and its port is set to 0, the automatic port allocation feature will also be used.
OUT CSocketAddr* pEffectiveLocalAddress	On output, contains the effective local address where the socket is bound. May be NULL if the user does not care.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

Used to associate a local address to the socket. The user MUST call this method following a call to Create.

2.11.1.31.1.2 - IClientSocket::Connect Method

Connects the socket to a remote address.

C++

```
virtual mxt_result Connect(IN const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pPeerAddress	Remote address where the socket is to be connected.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to connect the socket to a remote address. Once a socket is connected, Send (see page 640) and Recv (see page 638) can be used to exchange data with the remote peer.

2.11.1.32 - IIoSocket Class

Interface defining the methods used by a socket to send and receive data.

Class Hierarchy**C++**

```
class IIoSocket : public ISocket;
```

Description

The IIoSocket interface defines the methods accessible on a socket that can be used to send and receive data.

Location

Network/IIoSocket.h

See Also

IClientSocket (see page 635), IServerSocket (see page 644), ISocket (see page 646)

Methods

Method	Description
• A GetPeerAddress (see page 638)	Retrieves the remote address to which the socket is connected.
• A Recv (see page 638)	Receives data from a connected socket.
• A RecvFrom (see page 639)	Receives data from a non-connected socket.
• A Send (see page 640)	Sends data on a connected socket.
• A SendTo (see page 641)	Sends data on a non-connected socket.

ISocket Class

ISocket Class	Description
• A Close (see page 647)	Closes a socket.
• A GetAddressFamily (see page 647)	Retrieves the address family of this socket.

•  GetLocalAddress (see page 647)	Retrieves the local address to which the socket is bound.
•  GetProtocolFamily (see page 648)	Retrieves the protocol family of this socket.
•  GetSocketType (see page 648)	Retrieves information about the socket and its transport.
•  Release (see page 648)	Deletes this socket.
•  Set8021QUserPriority (see page 649)	Configures the 802.1Q user priority for this socket.
•  SetTos (see page 649)	Configures the type of service (TOS (see page 39)) associated with this socket.

Legend

	Method
	abstract

2.11.1.32.1 - Methods

2.11.1.32.1.1 - **IloSocket::GetPeerAddress** Method

Retrieves the remote address to which the socket is connected.

C++

```
virtual mxt_result GetPeerAddress(OUT CSocketAddr* pPeerAddress) const = 0;
```

Parameters

Parameters	Description
OUT CSocketAddr* pPeerAddress	OUT (see page 39) parameter used to receive the address.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address to which the socket is currently connected. It can return an error for unconnected sockets.

2.11.1.32.1.2 - Recv

2.11.1.32.1.2.1 - **IloSocket::Recv** Method

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT CBlob* pData) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pData	Blob object that will be filled with the data received on the socket.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See **GetSocketErrorId** (see page 656) for possible return values.

Description

This method is used to receive data from a connected socket.

The maximum number of bytes that can be received is defined by the blob's capacity. The user of the socket might want to resize the blob to a specific size before calling Recv, as its capacity will not be changed by this call.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- [IloSocket::Recv](#)
- [IloSocket::RecvFrom \(see page 639\)](#)

2.11.1.32.1.2.2 - IloSocket::Recv Method

Receives data from a connected socket.

C++

```
virtual mxt_result Recv(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puData	Pointer to a buffer where the received data is to be stored.
IN unsigned int uCapacity	The maximum size available in the buffer pointed to by puData.
OUT unsigned int* puSize	OUT (see page 39) parameter that contains the number of bytes actually written in the buffer.

Returns

- [resFE_INVALID_STATE](#)
- [resFE_INVALID_ARGUMENT](#)
- See [GetSocketErrorId \(see page 656\)](#) for possible return values.

Description

This method is used to receive data from a connected socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv.

See Also

- [IloSocket::Recv](#)
- [IloSocket::RecvFrom \(see page 639\)](#)

2.11.1.32.1.3 - RecvFrom**2.11.1.32.1.3.1 - IloSocket::RecvFrom Method**

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT CBlob* pData, OUT CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
OUT CBlob* pData	Blob object that will be filled with the data received on the socket.
OUT CSocketAddr* pPeerAddress	Address from which the data was received.

Returns

- [resFE_INVALID_STATE](#)
- [resFE_INVALID_ARGUMENT](#)
- See [GetSocketErrorId \(see page 656\)](#) for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in the Blob, no error will be returned. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv (see page 638).

See Also

- [IloSocket::RecvFrom](#)
- [IloSocket::Recv](#) (see page 638)

2.11.1.32.1.3.2 - IloSocket::RecvFrom Method

Receives data from a non-connected socket.

C++

```
virtual mxt_result RecvFrom(OUT uint8_t* puData, IN unsigned int uCapacity, OUT unsigned int* puSize, OUT CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
OUT uint8_t* puData	Pointer to a buffer where the received data is to be stored.
IN unsigned int uCapacity	The maximum size available in the buffer pointed to by puData.
OUT unsigned int* puSize	OUT (see page 39) parameter that contains the number of bytes actually written in the buffer.
OUT CSocketAddr* pPeerAddress	Address from which the data was received.

Returns

- [resFE_INVALID_STATE](#)
- [resFE_INVALID_ARGUMENT](#)
- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

This method is used to receive data from an unconnected socket. This should be called only when notified that there is data available on the socket.

When the received data is too big to fit in puData, no error will be returned, and puSize will be set to uCapacity. In UDP, the remaining data is discarded. In TCP, the remaining data is available by making another call to Recv (see page 638).

See Also

- [IloSocket::RecvFrom](#)
- [IloSocket::Recv](#) (see page 638)

2.11.1.32.1.4 - Send**2.11.1.32.1.4.1 - IloSocket::Send Method**

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const CBlob* pData, OUT unsigned int* puSizeSent) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Pointer to a blob object that contains the data to be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- [resFE_INVALID_STATE](#)
- [resFE_INVALID_ARGUMENT](#)
- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::Send
- IloSocket::SendTo (see page 641)

2.11.1.32.1.4.2 - IloSocket::Send Method

Sends data on a connected socket.

C++

```
virtual mxt_result Send(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to a buffer containing the data to send.
IN unsigned int uSize	The number of bytes in the buffer that must be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to send data on a connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- IloSocket::Send
- IloSocket::SendTo (see page 641)

2.11.1.32.1.5 - SendTo

2.11.1.32.1.5.1 - IloSocket::SendTo Method

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const CBlob* pData, OUT unsigned int* puSizeSent, IN const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const CBlob* pData	Pointer to a blob containing the data to be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
IN const CSocketAddr* pPeerAddress	Address to which the data must be sent.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- [IloSocket::SendTo](#)
- [IloSocket::Send](#) (see page 640)

2.11.1.32.1.5.2 - IloSocket::SendTo Method

Sends data on a non-connected socket.

C++

```
virtual mxt_result SendTo(IN const uint8_t* puData, IN unsigned int uSize, OUT unsigned int* puSizeSent, IN
const CSocketAddr* pPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puData	Pointer to a buffer containing the data to send.
IN unsigned int uSize	The number of bytes in the buffer that must be sent.
OUT unsigned int* puSizeSent	On output, will contain the number of bytes that were actually sent. The value returned in this parameter is not reliable if the method returns anything else than resS_OK.
IN const CSocketAddr* pPeerAddress	Address to which the data must be sent.

Returns

- [resFE_INVALID_STATE](#)
- [resFE_INVALID_ARGUMENT](#)
- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

This method is used to send data on a non-connected socket.

An application MUST not rely on the value of puSizeSent when this method returns anything else than resS_OK.

See Also

- [IloSocket::SendTo](#)
- [IloSocket::Send](#) (see page 640)

2.11.1.33 - IPolledRequestStatusMgr Class Symbian OS ONLY

Interface used for CPollRequestStatus (see page 536).

Class Hierarchy

```
IPolledRequestStatusMgr
```

C++

```
class IPolledRequestStatusMgr;
```

Description

This interface is to be used with the class CPollRequestStatus (see page 536). When registering a request status to CPollRequestStatus (see page 536), the user must provide this type of interface. Then, on a subsequent call to CPollRequestStatus::Poll (see page 539), this allows the class CPollRequestStatus (see page 536) to call back the user who registered a TRequestStatus once the request completed.

Location

Network/IPolledRequestStatusMgr.h

See Also

[CPollRequestStatus](#) (see page 536)

Methods

Method	Description
•  EvPolledRequestStatusMgrEventDetected (see page 643)	Notifies the manager about newly detected request completion.

Legend

	Method
	abstract

2.11.1.33.1 - Methods**2.11.1.33.1.1 - IPolledRequestStatusMgr::EvPolledRequestStatusMgrEventDetected Method**

Notifies the manager about newly detected request completion.

C++

```
virtual void EvPolledRequestStatusMgrEventDetected(IN TRequestStatus* pRequestStatus, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for the completed request.
IN mxt_opaque opq	The opaque value provided at registration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is called when completion of a request is detected.

Notes

the parameter pRequestStatus is there for flexibility. It gives the user the possibility to use the same IPolledRequestStatusMgr (see page 642) for a group of request status handles.

See Also

Network/CPollRequestStatus.h

2.11.1.34 - IPolledSocketMgr Class

Interface used for CPollSocket (see page 540).

Class Hierarchy



C++

```
class IPolledSocketMgr;
```

Description

This interface is to be used with the class CPollSocket (see page 540). When registering a socket to CPollSocket (see page 540), the user must provide this type of interface. Then, on a subsequent call to CPollSocket::Poll (see page 543), this allows the class CPollSocket (see page 540) to call back the user who registered a socket when there are events detected on the socket.

Location

Network/IPolledSocketMgr.h

See Also

CPollSocket (see page 540)

Methods

Method	Description
  EvPolledSocketMgrEventDetected (see page 644)	Notifies the manager about newly detected events on a socket.

Legend

	Method
	abstract

2.11.1.34.1 - Methods**2.11.1.34.1.1 - IPolledSocketMgr::EvPolledSocketMgrEventDetected Method**

Notifies the manager about newly detected events on a socket.

C++

```
virtual void EvPolledSocketMgrEventDetected(IN mxt_hSocket hSocket, IN unsigned int uEvents, IN mxt_opaque opq)
= 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle that has data ready to receive.
IN unsigned int uEvents	A bit field that indicates which events were detected.
IN mxt_opaque opq	The opaque value provided at registration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is called when at least one event was detected on a socket. Three different events may be detected: readability, writability and in exception.

Notes

the parameter hSocket is there for flexibility. It gives the user the possibility to use the same IPolledSocketMgr (see page 643) for a group of socket handles.

See Also

Network/CPollSocket.h

2.11.1.35 - IServerSocket Class

Interface defining the methods needed for a server socket.

Class Hierarchy**C++**

```
class IServerSocket : public ISocket;
```

Description

The IServerSocket interface defines the methods needed for a server type of socket. Note that this interface is available only to connection-oriented sockets types.

Location

Network/IServerSocket.h

See Also

IClientSocket (see page 635), IIoSocket (see page 637), ISocket (see page 646)

Methods

Method	Description
  Accept (see page 645)	Accepts an incoming connection attempt.
  Bind (see page 645)	Binds the socket to a local address.

• A Listen (see page 646)	Lists on the socket for incoming connection attempts.
---------------------------	---

ISocket Class

ISocket Class	Description
• A Close (see page 647)	Closes a socket.
• A GetAddressFamily (see page 647)	Retrieves the address family of this socket.
• A GetLocalAddress (see page 647)	Retrieves the local address to which the socket is bound.
• A GetProtocolFamily (see page 648)	Retrieves the protocol family of this socket.
• A GetSocketType (see page 648)	Retrieves information about the socket and its transport.
• A Release (see page 648)	Deletes this socket.
• A Set8021QUserPriority (see page 649)	Configures the 802.1Q user priority for this socket.
• A SetTos (see page 649)	Configures the type of service (TOS (see page 39) associated with this socket.

Legend

• A	Method
A	abstract

2.11.1.35.1 - Methods

2.11.1.35.1.1 - IServerSocket::Accept Method

Accepts an incoming connection attempt.

C++

```
virtual mxt_result Accept(OUT GO IIoSocket** ppIoSocket) = 0;
```

Parameters

Parameters	Description
OUT GO IIoSocket** ppIoSocket	OUT (see page 39) parameter used to receive the pointer to a IIoSocket (see page 637) representing the accepted connection. Note that the caller will receive the ownership of this pointer. This means that it will be the responsibility of the caller to delete this pointer by calling Release (see page 648) on it (unless the ownership of this pointer is given to some other entity).

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

Accepts an incoming connection attempt on a listening connection-oriented socket.

Please take note of the ownership note given in the parameter description.

See Also

Listen (see page 646)

2.11.1.35.1.2 - IServerSocket::Bind Method

Binds the socket to a local address.

C++

```
virtual mxt_result Bind(IN const CSocketAddr* pLocalAddress, OUT CSocketAddr* pEffectiveLocalAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr* pLocalAddress	Contains the local address where to bind. May be NULL if the user does not care about the interface or the local port.
OUT CSocketAddr* pEffectiveLocalAddress	On output, contains the effective local address where the socket is bound. May be NULL if the user does not care.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

Used to associate a local address to the socket. The user MUST call this method following a call to Create.

2.11.1.35.1.3 - *I ServerSocket::Listen* Method

Listens on the socket for incoming connection attempts.

C++

```
virtual mxt_result Listen(IN unsigned int uMaxPendingConnection = 5) = 0;
```

Parameters

Parameters	Description
IN unsigned int uMaxPendingConnection = 5	Maximum number of pending connections accepted.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

Sets the socket to listen for incoming connection attempts. Once a connection attempt is detected, Accept (see page 645) must be called in order to fully establish the connection.

See Also

Accept (see page 645)

2.11.1.36 - *ISocket* Class

Interface offering common functionalities to all types of sockets.

Class Hierarchy

```
ISocket
```

C++

```
class ISocket;
```

Description

The ISocket interface is the base interface shared by all socket classes. This interface offers the common functionality associated to all types of sockets.

Location

Network/ISocket.h

See Also

IClientSocket (see page 635), IServerSocket (see page 644), IIOSocket (see page 637)

Methods

Method	Description
• A Close (see page 647)	Closes a socket.
• A GetAddressFamily (see page 647)	Retrieves the address family of this socket.
• A GetLocalAddress (see page 647)	Retrieves the local address to which the socket is bound.
• A GetProtocolFamily (see page 648)	Retrieves the protocol family of this socket.
• A GetSocketType (see page 648)	Retrieves information about the socket and its transport.
• A Release (see page 648)	Deletes this socket.

  Set8021QUserPriority (see page 649)	Configures the 802.1Q user priority for this socket.
  SetTos (see page 649)	Configures the type of service (TOS (see page 39)) associated with this socket.

Legend

	Method
	abstract

2.11.1.36.1 - Methods

2.11.1.36.1.1 - ISocket::Close Method

Closes a socket.

C++

```
virtual mxt_result Close(IN ECloseBehavior eBehavior) = 0;
```

Parameters

Parameters	Description
IN ECloseBehavior eBehavior	How the socket is to be closed, as defined by the ECloseBehavior (see page 651) enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

This method is used to close a socket. This puts the socket into the state it was just after its allocation. After this call, all system resources associated to the socket must have been released. Note that a socket can be reused after it has been closed. In such a case, the socket options are all back to their default values. In other words, calling Close() on a socket has the effect of resetting the socket options to their default values after it is closed.

2.11.1.36.1.2 - ISocket::GetAddressFamily Method

Retrieves the address family of this socket.

C++

```
virtual mxt_result GetAddressFamily(OUT CSocketAddr::EAddressFamily* peAddressFamily) const = 0;
```

Parameters

Parameters	Description
OUT CSocketAddr::EAddressFamily* peAddressFamily	OUT (see page 39) parameter that receives the address family, as defined in the CSocketAddr::EAddressFamily (see page 651) enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address family associated to the socket.

2.11.1.36.1.3 - ISocket::GetLocalAddress Method

Retrieves the local address to which the socket is bound.

C++

```
virtual mxt_result GetLocalAddress(OUT CSocketAddr* pLocalAddress) const = 0;
```

Parameters

Parameters	Description
OUT CSocketAddr* pLocalAddress	OUT (see page 39) parameter that will contain the address and port information.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the address and port to which the socket has been bound. It will return an error until the socket is bound to an interface with the Bind function.

2.11.1.36.1.4 - **ISocket::GetProtocolFamily** Method Deprecated since v2.1.4

Retrieves the protocol family of this socket.

C++

```
virtual mxt_result GetProtocolFamily(OUT EProtocolFamily* peProtocolFamily) const = 0;
```

Parameters

Parameters	Description
OUT EProtocolFamily* peProtocolFamily	OUT (see page 39) parameter that receives the protocol family, as defined in the EProtocolFamily (see page 651) enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve the protocol family associated to the socket. This method is deprecated and GetAddressFamily (see page 647) should be used instead.

2.11.1.36.1.5 - **ISocket::GetSocketType** Method

Retrieves information about the socket and its transport.

C++

```
virtual mxt_result GetSocketType(OUT ESocketType* peSocketType) const = 0;
```

Parameters

Parameters	Description
OUT ESocketType* peSocketType	OUT (see page 39) parameter that receives the socket type information, as defined in the ESocketType (see page 652) enumeration.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- resS_OK

Description

This method is used to retrieve information about this socket and its transport.

2.11.1.36.1.6 - **ISocket::Release** Method

Deletes this socket.

C++

```
virtual uint32_t Release() = 0;
```

Returns

Always zero.

Description

This method is used to delete a socket. This has become necessary in order to be able to delete the socket from one of its interfaces (instead of directly calling the class that implements these interfaces). This is called by the user of the socket instead of the delete operator.

Note that this can be called if the socket is not closed yet. In such a case, the socket will be closed automatically before the socket is deleted.

The return value is always zero. We eventually plan to implement reference counting on the sockets, and the reference count number will be returned instead of zero when calling Release.

2.11.1.36.1.7 - ISocket::Set8021QUserPriority Method

Configures the 802.1Q user priority for this socket.

C++

```
virtual mxt_result Set8021QUserPriority(IN bool bEnable, IN uint8_t uUserPriority) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	Enables 802.1Q on this socket.
IN uint8_t uUserPriority	The user priority, from 0 to 7.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets the 802.1Q user priority for this socket. Note that this is not supported on all operating systems. This works correctly with the internal linux distribution of Mediatrix and with the modified Windriver IP stack.

This method is not used by MxSF.

2.11.1.36.1.8 - ISocket::SetTos Method

Configures the type of service (TOS (see page 39)) associated with this socket.

C++

```
virtual mxt_result SetTos(IN uint8_t uTos) = 0;
```

Parameters

Parameters	Description
IN uint8_t uTos	TOS (see page 39) byte for this socket.

Returns

- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT
- See GetSocketErrorId (see page 656) for possible return values.

Description

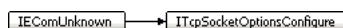
Configures the socket to include a specific TOS (see page 39) byte on the data it sends.

This method is not used by MxSF.

2.11.1.37 - ITcpSocketOptionsConfigure Class

Interface allowing the client to provide the necessary configuration information required by objects engaged in setting TCP socket options.

Class Hierarchy



C++

```
class ITcpSocketOptionsConfigure : public IEComUnknown;
```

Description

This interface allows the client to provide the necessary configuration information required by objects engaged in setting TCP Socket options.

Location

Network/ITcpSocketOptionsConfigure.h

Methods

Method	Description
• A ApplyOptions (see page 650)	Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.11.1.37.1 - Methods

2.11.1.37.1.1 - ITcpSocketOptionsConfigure::ApplyOptions Method

Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

C++

```
virtual mxt_result ApplyOptions(IN IAsyncSocket* pIAsyncSocket) = 0;
```

Returns

resS_OK: The socket options have been successfully set. resFE_MITOSFW_ECOM_NOINTERFACE

Description

Apply the options to be used, stored in the instance of this interface, on the socket provided as argument.

2.11.2 - Enumerations

This section documents the enumerations of the Sources/Network folder.

Enumerations

Enumeration	Description
EAddressFamily (see page 651)	Address families supported.
EStandardAddress (see page 651)	Standard IP addresses.
ECloseBehavior (see page 651)	Possible behaviours when closing.
EProtocolFamily (see page 651)	Possible protocol families.
ESocketType (see page 652)	This defines the type of transport associated with a socket.

2.11.2.1 - CSocketAddr::EAddressFamily Enumeration

Address families supported.

C++

```
enum EAddressFamily {
    eINET,
    eINET6
};
```

Members

Members	Description
eINET	IPv4 address family.
eINET6	IPv6 address family.

2.11.2.2 - CSocketAddr::EStandardAddress Enumeration

Standard IP addresses.

C++

```
enum EStandardAddress {
    eINET_ANY,
    eINET_BROADCAST,
    eINET_LOOPBACK,
    eINET_NONE,
    eINET6_ANY,
    eINET6_LOOPBACK
};
```

Members

Members	Description
eINET_ANY	IPv4 any address (0.0.0.0).
eINET_BROADCAST	IPv4 broadcast address (255.255.255.255).
eINET_LOOPBACK	IPv4 loopback address (127.0.0.1).
eINET_NONE	IPv4 none address (255.255.255.255).
eINET6_ANY	IPv6 any address (::). MXD_IPV6_ENABLE_SUPPORT (see page 294) must be defined to access this value.
eINET6_LOOPBACK	IPv6 loopback address (::1). MXD_IPV6_ENABLE_SUPPORT (see page 294) must be defined to access this value.

2.11.2.3 - ISocket::ECloseBehavior Enumeration

Possible behaviours when closing.

C++

```
enum ECloseBehavior {
    eGRACEFUL,
    eFORCE
};
```

Description

Possible behaviours when closing.

Members

Members	Description
eGRACEFUL	The close operation returns immediately and the socket will be closed gracefully in the background. After the close operation no more receives and sends can be issued on the socket. The data in the send buffer is sent and followed by FIN.
eFORCE	The socket will be forcefully closed. This means that all the queued data that has not yet been sent will be discarded. A RST is sent to the peer and will cause a resFE_MITOSFW_NET_SOCKET_CONNECTIONLOST on the peer.

2.11.2.4 - ISocket::EProtocolFamily Enumeration

Possible protocol families.

C++

```
enum EProtocolFamily {
```

```

eINTERNET_V4,
eINTERNET_V6,
eUNKNOWN
};

```

Description

Possible protocol families.

Members

Members	Description
eINTERNET_V4	This defines the IPv4 protocol family.
eINTERNET_V6	This defines the IPv6 protocol family.
eUNKNOWN	Unknown protocol family.

2.11.2.5 - ISocket::ESocketType Enumeration

This defines the type of transport associated with a socket.

C++

```

enum ESocketType {
    eSTREAM,
    eDATAGRAM,
    eSEQPACKET
};

```

Description

This defines the type of transport associated with a socket.

Members

Members	Description
eSTREAM	This defines a two-way, sequenced, reliable, connection-based byte-stream. An example of this is TCP.
eDATAGRAM	This defines a connectionless, unreliable transport of messages of a fixed maximum length. An example of this is UDP.
eSEQPACKET	This defines a two-way, sequenced, reliable, connection-based transmission of datagrams. An example of this is SCTP.

2.11.3 - Functions

This section documents the functions of the Sources/Network folder.

Functions

Function	Description
GetAllLocalIpAddresses (See page 653)	Retrieves all local IPv4 and IPv6 addresses.
GetEnumUri (See page 653)	Generates an ENUM request for the specified services and E.164 number.
GetHostByAddr (See page 653)	Get the hosts name using its address.
GetHostByName (See page 654)	Get the hosts address using its name.
GetHostByName (See page 654)	Get the hosts address using its name.
GetLocalHostName (See page 654)	Get the name of the local host.
GetLocalIPAddr (See page 655)	Retrieves the local address used to connect to the specified remote peer.
GetLocalIPv4Addr (See page 655)	Retrieves the local IPv4 address used to connect to the specified remote peer.
GetLocalIPv6Addr (See page 655)	Retrieves the local IPv6 address used to connect to the specified remote peer.
GetNaptrRecord (See page 655)	Queries the domain for a list of NAPTR records.
GetSocketErrorId (See page 656)	Gets the socket error ID.
GetSocketErrorId (See page 656)	Gets the socket error ID.
GetSrvRecord (See page 656)	Queries hosts providing the desired service.
GetSrvRecord (See page 657)	Queries the domain for a list of service records.
MxCloseSocket (See page 657)	Closes a socket.
SetSockOpt8021QUserPriority (See page 657)	Sets the 8021Q user priority option.
SetSockOptAllowAnySource (See page 658)	Sets the allow any source option.
SetSockOptBlocking (See page 658)	Sets the socket operation to blocking/non-blocking.
SetSockOptBroadcast (See page 658)	Sets the socket's broadcast option.
SetSockOptIpv6UnicastHops (See page 659)	Sets the IPv6 outgoing unicast hop limit.
SetSockOptKeepAliveEnable (See page 659)	Sets the socket's broadcast option.
SetSockOptLinger (See page 659)	Sets the socket linger option.

SetSockOptNagle (see page 660)	Enables/disables the Nagle algorithm.
SetSockOptReceiveBufferSize (see page 660)	Sets the socket's receive buffer size option.
SetSockOptReuseAddress (see page 661)	Sets the reuse address option.
SetSockOptTos (see page 661)	Sets the socket's type of service (TOS (see page 39) byte).
SetSockOptTransmitBufferSize (see page 661)	Sets the socket's transmit buffer size option.
SetSockOptUdpChecksum (see page 662)	Enables/Disables UDP checksum.

2.11.3.1 - GetAllLocalIpAddresses Function

Retrieves all local IPv4 and IPv6 addresses.

C++

```
mxt_result GetAllLocalIpAddresses(OUT CVector<SLocalIpAddress>* pvecAllLocalIpAddresses);
```

Parameters

Parameters	Description
OUT CVector<SLocalIpAddress>* pvecAllLocalIpAddresses	Pointer to the CVector (see page 227) object where the interfaces information will be copied. MUST be a valid pointer to an empty vector.

Returns

- resS_OK: At least one interface was found.
- resFE_FAIL: Error occurred.

Description

Gets all the local IPv4 and IPv6 address of the local computer.

Notes

IPv6 address only returned if MXD_IPV6_ENABLE_SUPPORT (see page 294) is defined.

2.11.3.2 - GetEnumUri Function

Generates an ENUM request for the specified services and E.164 number.

C++

```
mxt_result GetEnumUri(IN const char* pszE164Number, IN const CList<SEnumService>& rlstSupportedServices, OUT CList<SEnumUri>& rlstEnumUri, IN const char* pszZone = NULL);
```

Parameters

Parameters	Description
IN const char* pszE164Number	A telephone number in the E.164 format.
IN const CList<SEnumService>& rlstSupportedServices	A CList (see page 170) of SEnumService (see page 662) structures containing the services supported by the caller of this method.
OUT CList<SEnumUri>& rlstEnumUri	A CList (see page 170) of SEnumUri (see page 662) structures that will contain, on success, the URIs that correspond to the supplied ENUM AUS.
IN const char* pszZone = NULL	A c-style string containing the zone suffix to append to the FQDN form of the AUS. If NULL, this parameter defaults to the default ENUM zone, "e164.arpa".

Returns

resS_OK: The ENUM request completed successfully.

Description

This method is used to generate ENUM requests. It will query the DNS for NAPTR records corresponding to the E.164 number specified, and return found URIs that support the services specified by the client. Should there be more than one URI corresponding to the E.164 number, the list of URIs returned will be ordered according to the order and priority field in their respective NAPTR records.

2.11.3.3 - GetHostByAddr Function

Get the hosts name using its address.

C++

```
bool GetHostByAddr(IN const CSocketAddr& rHostAddr, IN unsigned int uBufLen, OUT char* pszHostName);
```

Parameters

Parameters	Description
IN const CSocketAddr& rHostAddr	Address of the host to find the name.
IN unsigned int uBufLen	Length of the buffer provided to store the host name.
OUT char* pszHostName	Buffer to store the host name.

Returns

True if the host name has been resolved, false otherwise.

Description

Gets the name corresponding to the host address provided. If more than one name is associated, it returns the first record in the DNS response.

2.11.3.4 - GetHostByName Function

Get the hosts address using its name.

C++

```
bool GetHostByName(IN const char* pszHostName, OUT CList<CSocketAddr>& rlstHostAddr, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const char* pszHostName	String containing the name of the host to query.
OUT CList<CSocketAddr>& rlstHostAddr	Reference to a list that will contain the addresses of the host.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	Family of the address to return. Currently, IPv4 and IPv6 are supported.

Returns

True if the host name has been resolved, false otherwise.

Description

Gets the list of addresses corresponding to the host string provided. An example of a valid host name is www.m5t.com .

2.11.3.5 - GetHostByName Function

Get the hosts address using its name.

C++

```
bool GetHostByName(IN const char* pszHostName, OUT CSocketAddr& rHostAddr, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const char* pszHostName	String containing the name of the host to query.
OUT CSocketAddr& rHostAddr	Reference to a CSocketAddr (see page 545) that will contain the host address.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	Family of the address to return. Currently, only IPv4 is supported.

Returns

True if the host name has been resolved, false otherwise.

Description

Gets the address corresponding to the host string provided. An example of a valid host name is www.m5t.com .

2.11.3.6 - GetLocalHostName Function

Get the name of the local host.

C++

```
bool GetLocalHostName(IN unsigned int nNameLen, OUT char* pszName);
```

Parameters

Parameters	Description
IN unsigned int nNameLen	Length of the buffer provided to store the host name.

OUT `char* pszName`

Buffer to store the host name.

Returns

True if the host name has been retrieved successfully, false otherwise.

Description

Get the name of the local host.

2.11.3.7 - GetLocalIPAddr Function

Retrieves the local address used to connect to the specified remote peer.

C++

```
CSocketAddr GetLocalIPAddr(IN const CSocketAddr& rPeerAddress = "0.0.0.0");
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddress = "0.0.0.0"	Reference to a CSocketAddr (see page 545) containing the peer's address.

Returns

A CSocketAddr (see page 545) containing the local address

Description

Get the local IPv4 or IPv6 address that can communicate with the remote host provided.

2.11.3.8 - GetLocalIPv4Addr Function

Retrieves the local IPv4 address used to connect to the specified remote peer.

C++

```
CSocketAddr GetLocalIPv4Addr(IN const CSocketAddr& rPeerAddress = "0.0.0.0");
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddress = "0.0.0.0"	Reference to a CSocketAddr (see page 545) containing the peer's address.

Returns

A CSocketAddr (see page 545) containing the local address

Description

Get the local IPv4 address that can communicate with the remote host provided.

2.11.3.9 - GetLocalIPv6Addr Function

Retrieves the local IPv6 address used to connect to the specified remote peer.

C++

```
CSocketAddr GetLocalIPv6Addr(IN const CSocketAddr& rPeerAddress = CSocketAddr("::"));
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddress = CSocketAddr("::")	Reference to a CSocketAddr (see page 545) containing the peer's IPv6 address.

Returns

A CSocketAddr (see page 545) containing the local IPv6 address.

Description

Get the local IPv6 address that can communicate with the remote host provided.

2.11.3.10 - GetNaptrRecord Function

Queries the domain for a list of NAPTR records.

C++

```
bool GetNaptrRecord(IN const char* pszDomainName, OUT CList<SNaptrRecord>& rlstNaptrRecord, IN
CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const char* pszDomainName	Domain name to query.
OUT CList<SNaptrRecord>& rlstNaptrRecord	CList (see page 170) of SNaptrRecord to fill with the answers.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	Address family of the addresses to put in rlstNaptrRecord.

Returns

True if the NAPTR query succeeded, false otherwise.

Description

Issues a NAPTR query to the DNS server and fills rlstNaptrRecord with the answers.

2.11.3.11 - GetSocketErrorId Function

Gets the socket error ID.

C++

```
mxt_result GetSocketErrorId(int nOsError);
mxt_result GetSocketErrorId();
```

Returns

- resS_OK
- resFE_MITOSFW_NET_SOCK_INITIALIZE
- resFE_MITOSFW_NET_SOCK_RESOURCES
- resFE_INVALID_ARGUMENT
- resFE_MITOSFW_NET_SOCK_ADDRESSNOTACCESSIBLE
- resFE_MITOSFW_NET_SOCK_WOULDBLOCK
- resFE_MITOSFW_NET_SOCK_CONNECTIONLOST
- resFE_MITOSFW_NET_SOCK_MSGSIZE
- resFE_MITOSFW_NET_SOCK_HOST_UNREACHABLE
- resFE_MITOSFW_NET_SOCK_CONNREFUSED
- resFE_MITOSFW_NET_SOCK_NOPROTOOPT
- resFE_MITOSFW_NET_SOCK_UNHANDLED

Description

Will map implementation specific error codes to mxt_result (see page 92) error id.

Example

This shows how to do GetErrorId().

```
mxt_result res = resS_OK;
if ( ( sd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) ) == -1 )
{
    res = GetSocketErrorId();
}
```

2.11.3.12 - GetSrvRecord Function

Queries hosts providing the desired service.

C++

```
bool GetSrvRecord(IN const char* pszService, IN const char* pszProtocol, IN const char* pszDomainName, OUT
CList<SSrvRecord>& rlstSrvRecord, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const char* pszService	Service for which to issue a SRV query.
IN const char* pszProtocol	Protocol for which to issue a SRV query.
IN const char* pszDomainName	Domain name to query.
OUT CList<SSrvRecord>& rlstSrvRecord	CList (see page 170) of SSrvRecord to fill with the answers.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	Address family of the addresses to put in rlstNaptrRecord.

Returns

True if the SRV query succeeded, false otherwise.

Description

Issues a SRV query to the DNS server and fills rlstSrvRecord with the answers.

Notes

Refer to RFC 1035 and RFC 2782 for a better understanding of this method

2.11.3.13 - GetSrvRecord Function

Queries the domain for a list of service records.

C++

```
bool GetSrvRecord(IN const char* pszSrvQueryName, OUT CList<SSrvRecord>& rlstSrvRecord, IN
CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const char* pszSrvQueryName	String to query the DNS server.
OUT CList<SSrvRecord>& rlstSrvRecord	CList (see page 170) of SSrvRecord to fill with the answers.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	Address family of the addresses to put in rlstNaptrRecord.

Returns

True if the SRV query succeeded, false otherwise.

Description

Issues a SRV query to the DNS server and fills rlstSrvRecord with the answers.

Notes

Refer to RFC 1035 and RFC 2782 for a better understanding of this method

2.11.3.14 - MxCloseSocket Function

Closes a socket.

C++

```
int MxCloseSocket(IN mxt_hSocket hSocket);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	Handle to a socket.

Returns

Returns 0 on success, -1 on error.

Description

Closes a socket handle.

2.11.3.15 - SetSockOpt8021QUserPriority Function

Sets the 8021Q user priority option.

C++

```
mxt_result SetSockOpt8021QUserPriority(IN mxt_hSocket hSocket, IN bool bEnable, IN uint8_t uUserPriority);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option.
IN bool bEnable	Determines whether the option is enabled or not.
IN uint8_t uUserPriority	The value to use in the 8021Q user priority field.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which simply enables/disables utilisation of the 8021Q user priority. If the option is enabled, then the 8021Q user priority field will be set to uUserPriority.

Notes: This option is proprietary. It is only supported on modified linux and VxWorks kernels. For any other target, the execution of this method will output an error trace before returning with resS_OK.

2.11.3.16 - SetSockOptAllowAnySource Function

Sets the allow any source option.

C++

```
mxt_result SetSockOptAllowAnySource(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option.
IN bool bEnable	Determines if the option is enable (true) or not (false).

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which simply enables/disables the socket from being capable of receiving from multiple sources even if the socket is connected. Usually, connected sockets will prevent receiving from any other source IP address and port combination than the one the socket was connected to.

Notes: This option is proprietary. It is only supported on modified linux kernels. For any other target, the execution of this method will output an error trace before returning with resS_OK.

2.11.3.17 - SetSockOptBlocking Function

Sets the socket operation to blocking/non-blocking.

C++

```
mxt_result SetSockOptBlocking(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN bool bEnable	Determines if the socket operation mode is blocking (true) or not (false)

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines the blocking mode operation of the socket. When bEnable is true, all socket operations will block the calling process. If bEnable is false, then the calling process will not block and the operation will return resFE_MITOSFW_NET_SOCK_WOULDBLOCK if the call to that operation would have blocked. The default value of this option is true.

2.11.3.18 - SetSockOptBroadcast Function

Sets the socket's broadcast option.

C++

```
mxt_result SetSockOptBroadcast(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN bool bEnable	Determines if the socket will send broadcast or not. When bEnable is true broadcast are sent otherwise, unicast are sent.

Returns

- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

This method sets a socket level option which determines if broadcast or unicast messages are going to be sent. The default value of this option is false.

2.11.3.19 - SetSockOptIpv6UnicastHops Function

Sets the IPv6 outgoing unicast hop limit.

C++

```
mxt_result SetSockOptIpv6UnicastHops(IN mxt_hSocket hSocket, IN int nHops);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN int nHops	Number of hops.

Returns

- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

Sets the IPv6 outgoing unicast hop limit.

2.11.3.20 - SetSockOptKeepAliveEnable Function

Sets the socket's broadcast option.

C++

```
mxt_result SetSockOptKeepAliveEnable(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option.
IN bool bEnable	If true, enable the TCP keep-alive messages on the socket.

Returns

- See [GetSocketErrorId](#) (see page 656) for possible return values.

Description

Enables the TCP connection's option to send keep-alive messages on idle TCP connections, using the operating system's default settings.

2.11.3.21 - SetSockOptLinger Function

Sets the socket linger option.

C++

```
mxt_result SetSockOptLinger(IN mxt_hSocket hSocket, IN bool bOnOff, IN unsigned int uLinger);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option

IN bool bOnOff	Determines if the socket will be closed gracefully or not depending of the value of nLinger.
IN unsigned int uLinger	Determines if the socket will be closed gracefully or not. If nLinger is 0 the socket will be closed forcefully. If it is a positive value it will wait until either the socket is closed or the nLinger timeout expires.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines how the socket is closed. Three scenarios are possible:

1. If bOnOff is false, nLinger will be ignored. This is the default.

The close operation return immediately and the socket will be closed gracefully in the background. After the close operation no more receives and sends can be issued on the socket. The data in the send buffer is sent and followed by FIN.

2. If bOnOff is true and nLinger is zero, the close operation will be

forceful. The data in the send buffer is flushed and a RST is sent to the peer. This will cause a resFE_MITOSFW_NET_SOCKET_CONNECTIONLOST on the peer.

3. If bOnOff is true and nLinger is positive the close will return

when the data in the send buffer is sent and acknowledged or when the nLinger timeout (interpretation of the timeout is implementation dependent) is expired. A non-blocking socket will return right away but the result is implementation dependent and unclear, so this option should not be used on non-blocking sockets.

2.11.3.22 - SetSockOptNagle Function

Enables/disables the Nagle algorithm.

C++

```
mxt_result SetSockOptNagle(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN bool bEnable	Enables if true / Disables if false

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines if the Nagle algorithm will be used or not. This applies to TCP only.

Notes: This option is target dependent. If it is compiled for a target that does not support the option like VxWorks 6.5, the execution of this method will output an error trace before returning with resS_OK.

2.11.3.23 - SetSockOptReceiveBufferSize Function

Sets the socket's receive buffer size option.

C++

```
mxt_result SetSockOptReceiveBufferSize(IN mxt_hSocket hSocket, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN unsigned int uSize	The size of the receive buffer.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines the maximum socket receive buffer in bytes. Usually the default value of this option is adequate. If you use this option, you better know what you are doing.

Notes: This option is target dependant. If it is compiled for a target that does not support the option like Nucleus, the execution of this method will output an error trace before returning with resS_OK .

2.11.3.24 - SetSockOptReuseAddress Function

Sets the reuse address option.

C++

```
mxt_result SetSockOptReuseAddress(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN bool bEnable	Determines if the option is enabled (true) or not (false)

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which simply enables/disables the socket from being capable of binding more than once at the same local address. Note this option will apply only if called before Bind().

2.11.3.25 - SetSockOptTos Function

Sets the socket's type of service (TOS (see page 39) byte).

C++

```
mxt_result SetSockOptTos(IN mxt_hSocket hSocket, IN uint8_t uTos);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN uint8_t uTos	The value of the TOS (see page 39) byte field for this socket.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which simply sets the TOS (see page 39) byte field in the IP protocol header.

For this option to work under Windows 2000, XP, and 2003 Server the value **DisableUserTOSSetting** must be set in the registry. It is located in the following key: HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters The value is a REG_DWORD that must be set to 0 in order for the TOS (see page 39) flags to be set in the IP header. On Windows 2000 and some older builds of Windows XP the value must be added. Windows must be rebooted for the changes to take effect. Additional information can be found here: <http://support.microsoft.com/default.aspx?scid=kb;en-us;248611>

Notes: This option is target dependant. If it is compiled for a target that does not support the option like Windows CE, the execution of this method will output an error trace before returning with resS_OK .

- In Windows CE 4.0, the function call succeeds but the TOS (see page 39) byte field in the IP protocol header will remain unchanged.
- Under Nucleus, this method sets the DSCP bytefield instead of the TOS (see page 39). It is impossible to set the TOS (see page 39) byte field.

2.11.3.26 - SetSockOptTransmitBufferSize Function

Sets the socket's transmit buffer size option.

C++

```
mxt_result SetSockOptTransmitBufferSize(IN mxt_hSocket hSocket, IN unsigned int uSize);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN unsigned int uSize	The size of the transmit buffer.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

This method sets a socket level option which determines the maximum socket transmit buffer in bytes. Usually, default value of this option is adequate. If you use this option, you better know what your are doing.

Notes: This option is target dependant. If it is compiled for a target that does not support the option like Nucleus, the execution of this method will output an error trace before returning with resS_OK.

2.11.3.27 - SetSockOptUdpChecksum Function

Enables/Disables UDP checksum.

C++

```
mxt_result SetSockOptUdpChecksum(IN mxt_hSocket hSocket, IN bool bEnable);
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The socket handle on which to apply the option
IN bool bEnable	bool to enable or disable this option.

Returns

- See GetSocketErrorId (see page 656) for possible return values.

Description

Enables/Disables the UDP checksum flag on the specified socket.

Notes: This option is target dependant. If it is compiled for a target that does not support the option like Nucleus, VxWorks, Windows and Windows CE the execution of this method will output an error trace before returning with resS_OK.

2.11.4 - Structures

This section documents the structures of the Sources/Network folder.

Structs

Struct	Description
SEnumService (see page 662)	Structure representing an enumservice and its supported sub-services.
SEnumUri (see page 662)	Structure representing an URI returned by an ENUM request.
SLocalIpAddress (see page 663)	Structure to store all the local IP addresses and their associated interface name for the local machine.

2.11.4.1 - SEnumService Struct

Structure representing an enumservice and its supported sub-services.

C++

```
struct SEnumService {
    CString m_strEnumService;
    CList<CString> m_lststrEnumSubTypes;
};
```

Members

Members	Description
CString m_strEnumService;	The enumservice.
CList<CString> m_lststrEnumSubTypes;	Sub-services supported by the enumservice.

2.11.4.2 - SEnumUri Struct

Structure representing an URI returned by an ENUM request.

C++

```
struct SEnumUri {
    CString m_strUri;
    CList<SEnumService> m_lststEnumServices;
};
```

Members

Members	Description
CString m_strUri;	CString (see page 126) representing the URI.
CList<SEnumService> m_lststEnumServices;	List of enumservices supported by the URI.

2.11.4.3 - SLocalIpAddress Struct

Structure to store all the local IP addresses and their associated interface name for the local machine.

C++

```
struct SLocalIpAddress {
    CSocketAddr m_addrLocal;
    CString m_strInterfaceName;
};
```

Description

This structure is used to store all the local IP addresses and their associated interface name for the local machine.

Members

Members	Description
CSocketAddr m_addrLocal;	A CSocketAddr (see page 545) to keep the IP address.
CString m_strInterfaceName;	A CString (see page 126) to store the interface name.

2.11.5 - Variables

This section documents the variables of the Sources/Network folder.

2.12 - PkI

This section documents the Sources/PkI folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 663)

2.12.1 - Classes

This section documents the classes of the Sources/PkI folder.

Classes

Class	Description
CAAlternateName (see page 664)	CAAlternateName is the base class for the CIssuerAlternateName (see page 715) and CSubjectAlternateName (see page 732) extensions.
CAuthorityKeyIdentifier (see page 668)	Class to handle the authority key identifier extensions certificates.
CBasicConstraints (see page 674)	Class to handle the Basic Constraints extensions inside certificates.
CCertificate (see page 677)	Class that manages and abstracts an X.509 certificate.
CCertificateChain (see page 691)	Class that manages an ordered list of certificates.
CCertificateChainValidation (see page 696)	Performs certificate chain validation.
CCertificateExtension (see page 699)	Class managing certificate extensions.
CCertificateIssuer (see page 704)	Class managing the certificate issuer related identifiers and values.
CCertificateSubject (see page 707)	Class that manages the certificate subject related identifiers and values.
CExtendedKeyUsage (see page 711)	Class to handle the extended key usage extensions.
CIssuerAlternateName (see page 715)	Class that manages the issuerAlternateName extension.
CKeyUsage (see page 718)	Class to handle the key usage extensions inside certificates.
CNetscapeCertificateType (see page 723)	Class to handle the Netscape certificates extensions certificates.
CPkcs12 (see page 728)	Class that manages the creation and parsing of PKCS#12 v1.0 formatted buffers.
CSubjectAlternateName (see page 732)	Class that manages the subjectAlternateName extension.
CSubjectKeyIdentifier (see page 735)	Class used to handle the subject key identifier extensions.

2.12.1.1 - CAAlternateName Class

CAAlternateName is the base class for the CIssuerAlternateName (see page 715) and CSubjectAlternateName (see page 732) extensions.

Class Hierarchy



C++

```
class CAAlternateName : public MXD_PKI_CALTERNATENAME_CLASSNAME;
```

Description

CAAlternateName is the base class for the CIssuerAlternateName (see page 715) and CSubjectAlternateName (see page 732) extensions. It is a container of names that may be of multiple types.

Location

Pki/CAAlternateName.h

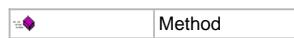
See Also

CIssuerAlternateName (see page 715), CSubjectAlternateName (see page 732), CCertificateExtension (see page 699)

Constructors

Constructor	Description
CAAlternateName (see page 665)	Constructor.

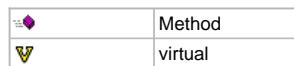
Legend



Destructors

Destructor	Description
~CAAlternateName (see page 665)	Destructor.

Legend



Operators

Operator	Description
!= (see page 667)	Different than operator.
= (see page 668)	Assignment operator.
== (see page 668)	Comparison operator.

Legend



Methods

Method	Description
GetDnsName (see page 665)	Gets the value of a name of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.
GetIpAddress (see page 666)	Gets the value of a name of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.
GetNameCount (see page 666)	Gets the number of name entries.
GetNameType (see page 666)	Gets the type of a name entry.
GetRfc822Name (see page 667)	Gets the value of the name of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.
GetUniformResourceIdentifier (see page 667)	Gets the value of a name of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

Legend



2.12.1.1.1 - Constructors

2.12.1.1.1.1 - CAAlternateName

2.12.1.1.1.1.1 - CAAlternateName::CAAlternateName Constructor

Constructor.

C++

```
CAAlternateName();
```

Description

Constructor.

2.12.1.1.1.1.2 - CAAlternateName::CAAlternateName Constructor

Copy constructor.

C++

```
CAAlternateName(IN const CAAlternateName& rAlternateName);
```

Parameters

Parameters	Description
IN const CAAlternateName& rAlternateName	Reference to the CAAlternateName to copy.

Description

Copy constructor.

2.12.1.1.1.1.3 - CAAlternateName::CAAlternateName Constructor

Copy constructor.

C++

```
CAAlternateName(IN const CAAlternateName* pAlternateName);
```

Parameters

Parameters	Description
IN const CAAlternateName* pAlternateName	Pointer to the CAAlternateName to copy.

Description

Copy constructor.

2.12.1.1.2 - Destructors

2.12.1.1.2.1 - CAAlternateName::~CAAlternateName Destructor

Destructor.

C++

```
virtual ~CAAlternateName();
```

Description

Destructor.

2.12.1.1.3 - Methods

2.12.1.1.3.1 - CAAlternateName::GetDnsName Method

Gets the value of a name of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.

C++

```
mxt_result GetDnsName(IN unsigned int uName, OUT CString* pstrDnsName) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the name.
OUT CString* pstrDnsName	Pointer to contain the DNS name.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the DNS name from the object.

2.12.1.1.3.2 - CAAlternateName::GetIpAddress Method

Gets the value of a name of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.

C++

```
mxt_result GetIpAddress(IN unsigned int uName, OUT CString* pstrIpAddress) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the name.
OUT CString* pstrIpAddress	Pointer to contain the IP address.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the IP address from the object.

2.12.1.1.3.3 - CAAlternateName::GetNameCount Method

Gets the number of name entries.

C++

```
mxt_result GetNameCount(OUT unsigned int* puNameCount) const;
```

Parameters

Parameters	Description
OUT unsigned int * puNameCount	Pointer to hold the name count.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the number of name items inside this object.

2.12.1.1.3.4 - CAAlternateName::GetNameType Method

Gets the type of a name entry.

C++

```
mxt_result GetNameType(IN unsigned int uName, OUT ENameType* peNameType) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the name.
OUT ENameType* peNameType	The type of the name at ulIndex.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the type of name stored at the specified index.

2.12.1.1.3.5 - CAlternateName::GetRfc822Name Method

Gets the value of the name of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.

C++

```
mxt_result GetRfc822Name(IN unsigned int uName, OUT CString* pstrRfc822Name) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the name.
OUT CString* pstrRfc822Name	Pointer to contain the RFC 822 name.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the RFC 822 name from the object.

2.12.1.1.3.6 - CAlternateName::GetUniformResourceIdentifier Method

Gets the value of a name of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

C++

```
mxt_result GetUniformResourceIdentifier(IN unsigned int uName, OUT CString* pstrUniformResourceIdentifier) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the name.
OUT CString* pstrUniformResourceIdentifier	Pointer to contain the uniform resource identifier.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the uniform resource identifier from the object.

2.12.1.1.4 - Operators

2.12.1.1.4.1 - CAlternateName::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CAlternateName& rAlternateName) const;
```

Parameters

Parameters	Description
IN const CAAlternateName& rAlternateName	Reference to the CAAlternateName (see page 664) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.1.4.2 - CAAlternateName::= Operator

Assignment operator.

C++

```
CAAlternateName& operator =(IN const CAAlternateName& rAlternateName);
```

Parameters

Parameters	Description
IN const CAAlternateName& rAlternateName	Reference to the CAAlternateName (see page 664) to copy.

Returns

Reference to the assigned CAAlternateName (see page 664).

Description

Assigns the right hand object to the left hand one.

2.12.1.1.4.3 - CAAlternateName::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CAAlternateName& rAlternateName) const;
```

Parameters

Parameters	Description
IN const CAAlternateName& rAlternateName	Reference to the CAAlternateName (see page 664) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.2 - CAAuthorityKeyIdentifier Class

Class to handle the authority key identifier extensions certificates.

Class Hierarchy



C++

```
class CAuthorityKeyIdentifier : public MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME;
```

Description

CAuthorityKeyIdentifier is the class to handle the authority key identifier extensions 2.5.29.35. They are used to identify the public key related to the private key used to sign a certificate.

Location

Pki/CAuthorityKeyIdentifier.h

See Also

CCertificateExtension (see page 699)

Constructors

Constructor	Description
CAuthorityKeyIdentifier (see page 669)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CAuthorityKeyIdentifier (see page 670)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 673)	Different than operator.
= (see page 673)	Assignment operator.
== (see page 673)	Comparison operator.

Legend

	Method
---	--------

Methods

Method	Description
GetCertificateIssuerCount (see page 670)	Gets the count of certificate issuers contained within this extension.
GetCertificateSerialNumber (see page 671)	Gets the serial number of the certificate.
GetDnsIssuer (see page 671)	Gets the value of an issuer of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.
GetIpAddressIssuer (see page 671)	Gets the value of an issuer of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.
GetIssuerType (see page 671)	Gets the type of issuer data.
GetKeyIdentifier (see page 672)	Gets the key identifier of the certificate.
GetRfc822NameIssuer (see page 672)	Gets the value of an issuer of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.
GetUniformResourceIdentifierIssuer (see page 672)	Gets the value of an issuer of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

Legend

	Method
---	--------

2.12.1.2.1 - Constructors

2.12.1.2.1.1 - CAuthorityKeyIdentifier

2.12.1.2.1.1.1 - CAuthorityKeyIdentifier::CAuthorityKeyIdentifier Constructor

Constructor.

C++

```
CAuthorityKeyIdentifier();
```

Description

Constructor.

2.12.1.2.1.1.2 - CAuthorityKeyIdentifier::CAuthorityKeyIdentifier Constructor

Copy constructor.

C++

```
CAuthorityKeyIdentifier(IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier);
```

Parameters

Parameters	Description
IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier	Reference to the object to copy.

Description

Constructor.

2.12.1.2.1.1.3 - CAuthorityKeyIdentifier::CAuthorityKeyIdentifier Constructor

Copy constructor.

C++

```
CAuthorityKeyIdentifier(IN const CAuthorityKeyIdentifier* pAuthorityKeyIdentifier);
```

Parameters

Parameters	Description
IN const CAuthorityKeyIdentifier* pAuthorityKeyIdentifier	Pointer to the object to copy.

Description

Constructor.

2.12.1.2.2 - Destructors

2.12.1.2.2.1 - CAuthorityKeyIdentifier::~CAuthorityKeyIdentifier Destructor

Destructor.

C++

```
virtual ~CAuthorityKeyIdentifier();
```

Description

Destructor.

2.12.1.2.3 - Methods

2.12.1.2.3.1 - CAuthorityKeyIdentifier::GetCertificateIssuerCount Method

Gets the count of certificate issuers contained within this extension.

C++

```
mxt_result GetCertificateIssuerCount(OUT unsigned int* puNameCount) const;
```

Parameters

Parameters	Description
OUT unsigned int* puNameCount	Number of issuers.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the number of certificate issuers that are present.

2.12.1.2.3.2 - CAuthorityKeyIdentifier::GetCertificateSerialNumber Method

Gets the serial number of the certificate.

C++

```
mxt_result GetCertificateSerialNumber(OUT CBlob* pSerialNumber) const;
```

Parameters

Parameters	Description
OUT CBlob* pSerialNumber	Pointer to a blob to hold the serial number.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the certificates serial number for this authority.

2.12.1.2.3.3 - CAuthorityKeyIdentifier::GetDnsIssuer Method

Gets the value of an issuer of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.

C++

```
mxt_result GetDnsIssuer(IN unsigned int uName, OUT CString* pstrDnsName) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.
OUT CString* pstrDnsName	Pointer to the CString (see page 126) object to contain the return value.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the DNS of the issuer at the specified index.

2.12.1.2.3.4 - CAuthorityKeyIdentifier::GetIpAddressIssuer Method

Gets the value of an issuer of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.

C++

```
mxt_result GetIpAddressIssuer(IN unsigned int uName, OUT CString* pstrIpAddress) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.
OUT CString* pstrIpAddress	Pointer to the CString (see page 126) object to contain the return value.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the IP address of the issuer at the specified index.

2.12.1.2.3.5 - CAuthorityKeyIdentifier::GetIssuerType Method

Gets the type of issuer data.

C++

```
mxt_result GetIssuerType(IN unsigned int uName, OUT ENameType* peNameType) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Index of the issuer to retrieve.
OUT ENameType* peNameType	The type of issuer contained at that index.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the type of issuer data at the specified index.

2.12.1.2.3.6 - CAuthorityKeyIdentifier::GetKeyIdentifier Method

Gets the key identifier of the certificate.

C++

```
mxt_result GetKeyIdentifier(OUT CBlob* pblobIdentifier) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobIdentifier	Pointer to the blob to hold the key identifier.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the key identifier for this authority.

2.12.1.2.3.7 - CAuthorityKeyIdentifier::GetRfc822NameIssuer Method

Gets the value of an issuer of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.

C++

```
mxt_result GetRfc822NameIssuer(IN unsigned int uName, OUT CString* pstrRfc822Name) const;
```

Parameters

Parameters	Description
IN unsigned int uName	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.
OUT CString* pstrRfc822Name	Pointer to the CString (see page 126) object to contain the return value.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the RFC 822 name of the issuer at the specified index.

2.12.1.2.3.8 - CAuthorityKeyIdentifier::GetUniformResourceIdentifierIssuer Method

Gets the value of an issuer of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

C++

```
mxt_result GetUniformResourceIdentifierIssuer(IN unsigned int uName, OUT CString* pstrUniformResourceIdentifier)
```

```
const;
```

Parameters

Parameters	Description
IN unsigned int uName	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.
OUT CString* pstrUniformResourceIdentifier	Pointer to the CString (see page 126) object to contain the return value.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the URI of the issuer at the specified index in the issuer.

2.12.1.2.4 - Operators

2.12.1.2.4.1 - CAuthorityKeyIdentifier::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier) const;
```

Parameters

Parameters	Description
IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.2.4.2 - CAuthorityKeyIdentifier::= Operator

Assignment operator.

C++

```
CAuthorityKeyIdentifier& operator =(IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier);
```

Parameters

Parameters	Description
IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.2.4.3 - CAuthorityKeyIdentifier::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier) const;
```

Parameters

Parameters	Description
IN const CAuthorityKeyIdentifier& rAuthorityKeyIdentifier	Reference to the CAuthorityKeyIdentifier (see page 668) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.3 - CBasicConstraints Class

Class to handle the Basic Constraints extensions inside certificates.

Class Hierarchy**C++**

```
class CBasicConstraints : public MXD_PKI_CBASICCONSTRAINTS_CLASSNAME;
```

Description

CBasicConstraints is the class to handle the Basic Constraints extensions inside certificates.

Notes: It may occur that some older certificates encountered have the obsolete ID for the basic constraints extension 2.5.29.10 instead of 2.5.29.19. If this is the case, the extension is considered as an unknown extension.

Location

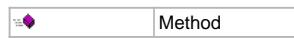
Pki/CBasicConstraints.h

See Also

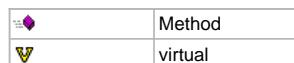
CCertificateExtension (see page 699)

Constructors

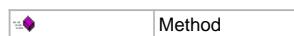
Constructor	Description
CBasicConstraints (see page 675)	Constructor.

Legend**Destructors**

Destructor	Description
~CBasicConstraints (see page 675)	Destructor.

Legend**Operators**

Operator	Description
!= (see page 676)	Different than operator.
= (see page 676)	Assignment operator.
== (see page 677)	Comparison operator.

Legend**Methods**

Method	Description
GetPathLengthConstraint (see page 675)	Gets the path length constraint.
IsACertificateAuthority (see page 676)	Verifies if it is a certificate authority.

Legend**2.12.1.3.1 - Constructors**

2.12.1.3.1.1 - CBasicConstraints

2.12.1.3.1.1.1 - CBasicConstraints::CBasicConstraints Constructor

Constructor.

C++

```
CBasicConstraints();
```

Description

Constructor.

2.12.1.3.1.1.2 - CBasicConstraints::CBasicConstraints Constructor

Copy constructor.

C++

```
CBasicConstraints(IN const CBasicConstraints& rBasicConstraints);
```

Parameters

Parameters	Description
IN const CBasicConstraints& rBasicConstraints	Reference to the object to copy.

Description

Constructor.

2.12.1.3.1.1.3 - CBasicConstraints::CBasicConstraints Constructor

Copy constructor.

C++

```
CBasicConstraints(IN const CBasicConstraints* pBasicConstraints);
```

Parameters

Parameters	Description
IN const CBasicConstraints* pBasicConstraints	Pointer to the object to copy.

Description

Constructor.

2.12.1.3.2 - Destructors

2.12.1.3.2.1 - CBasicConstraints::~CBasicConstraints Destructor

Destructor.

C++

```
virtual ~CBasicConstraints();
```

Description

Destructor.

2.12.1.3.3 - Methods

2.12.1.3.3.1 - CBasicConstraints::GetPathLengthConstraint Method

Gets the path length constraint.

C++

```
mxt_result GetPathLengthConstraint(OUT unsigned int* puPathLengthConstraint) const;
```

Parameters

Parameters	Description
OUT unsigned int * puPathLengthConstraint	Pointer to contain the path length constraint.

Returns

- **resFE_FAIL**
- **resFE_INVALID_ARGUMENT**
- **resSI_FALSE**: The Path Length Constraint does not exist.
- **resSI_TRUE**: The Path Length Constraint exists.

Description

Gets the path length constraint. The path length constraint is the maximum number of certificate authorities that can be included above this certificate in the certificate path.

2.12.1.3.3.2 - CBasicConstraints::IsACertificateAuthority Method

Verifies if it is a certificate authority.

C++

```
mxt_result IsACertificateAuthority() const;
```

Returns

- **resFE_FAIL**
- **resFE_INVALID_STATE**
- **resSI_FALSE**
- **resSI_TRUE**

Description

Verifies if the certificate has a certificate authority extension.

2.12.1.3.4 - Operators**2.12.1.3.4.1 - CBasicConstraints::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CBasicConstraints& rBasicConstraints) const;
```

Parameters

Parameters	Description
IN const CBasicConstraints& rBasicConstraints	Reference to the CBasicConstraints (see page 674) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.3.4.2 - CBasicConstraints::= Operator

Assignment operator.

C++

```
CBasicConstraints& operator =(IN const CBasicConstraints& rBasicConstraints);
```

Parameters

Parameters	Description
IN const CBasicConstraints& rBasicConstraints	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.3.4.3 - CBasicConstraints::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CBasicConstraints& rBasicConstraints) const;
```

Parameters

Parameters	Description
IN const CBasicConstraints& rBasicConstraints	Reference to the CBasicConstraints (see page 674) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.4 - CCertificate Class

Class that manages and abstracts an X.509 certificate.

Class Hierarchy**C++**

```
class CCertificate : public MXD_PKI_CCERTIFICATE_CLASSNAME;
```

Description

CCertificate is the class that manages and abstracts an X.509 certificate.

An X.509 certificate is used to certify the identity of an entity.

Location

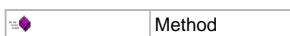
Pki/CCertificate.h

See Also

CCertificateChain (see page 691)

Constructors

Constructor	Description
CCertificate (see page 678)	Constructor.

Legend**Destructors**

Destructor	Description
~CCertificate (see page 679)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
<code>!=</code> (see page 690)	Different than operator.
<code>=</code> (see page 690)	Assignment operator.
<code>==</code> (see page 690)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
<code>DisplayCertificate</code> (see page 679)	Displays certificate information.
<code>GetAuthorityKeyIdentifierExtension</code> (see page 680)	Gets the Authority Key Identifier extension.
<code>GetBasicConstraintsExtension</code> (see page 680)	Gets the Basic Constraints extension.
<code>GetExtendedKeyUsageExtension</code> (see page 680)	Gets the Extended Key Usage certificate extension.
<code>GetExtension</code> (see page 681)	Gets the certificate extension information by index.
<code>GetExtensionByType</code> (see page 681)	Gets the certificate extension corresponding to a specified type.
<code>GetExtensionCount</code> (see page 681)	Gets the number of extensions added to this certificate.
<code>GetIssuer</code> (see page 682)	Gets the certificate issuer.
<code>GetIssuerAlternateNameExtension</code> (see page 682)	Gets the Issuer Alternate Name extension.
<code>GetKeyUsageExtension</code> (see page 682)	Gets the Key Usage certificate extension.
<code>GetNetscapeCertificateTypeExtension</code> (see page 682)	Gets the Netscape Certificate Type extension.
<code>GetNotAfterTime</code> (see page 683)	Gets the time after which the certificate is considered as invalid.
<code>GetNotBeforeTime</code> (see page 683)	Gets the time before which the certificate is considered as invalid.
<code>GetPrivateKey</code> (see page 683)	Gets a private key associated with this X.509 certificate.
<code>GetPublicKey</code> (see page 684)	Gets the public key associated with this certificate.
<code>GetSerialNumber</code> (see page 684)	Gets the certificate serial number.
<code>GetSignature</code> (see page 684)	Gets the signature and signature algorithm for this certificate.
<code>GetSubject</code> (see page 685)	Gets the certificate subject.
<code>GetSubjectAlternateNameExtension</code> (see page 685)	Gets the Subject Alternate Name extension.
<code>GetSubjectKeyIdentifierExtension</code> (see page 685)	Gets the Subject Key Identifier extension.
<code>GetVersion</code> (see page 686)	Gets the certificate version.
<code>IsIssuedBy</code> (see page 686)	Verifies whether a certificate has been issued by another one.
<code>IsSelfIssued</code> (see page 686)	Verifies whether a certificate is self-issued.
<code>Restore</code> (see page 687)	Reads a certificate (DER- or PEM-encoded) from a blob.
<code>RestoreDer</code> (see page 687)	Reads a DER-encoded certificate from a blob.
<code>RestorePem</code> (see page 687)	Reads a PEM-encoded certificate from a blob.
<code>SetPrivateKey</code> (see page 688)	Associates a private key with this X.509 certificate.
<code>Store</code> (see page 688)	Stores the content of this CCertificate into a blob, with a specified encoding.
<code>StoreDer</code> (see page 688)	Stores the content of this certificate in DER format into a blob.
<code>StorePem</code> (see page 689)	Stores the content of this certificate in PEM format into a blob.
<code>VerifySignature</code> (see page 689)	Validates if the certificate has been properly signed.

Legend

	Method
--	--------

2.12.1.4.1 - Constructors

2.12.1.4.1.1 - CCertificate

2.12.1.4.1.1.1 - CCertificate::CCertificate Constructor

Constructor.

C++

```
CCertificate();
```

Description

Constructor.

2.12.1.4.1.1.2 - CCertificate::CCertificate Constructor

Copy constructor.

C++

```
CCertificate(IN const CCertificate& rCertificate);
```

Parameters

Parameters	Description
IN const CCertificate& rCertificate	Reference to the CCertificate to copy.

Description

Copy constructor.

2.12.1.4.1.1.3 - CCertificate::CCertificate Constructor

Copy constructor.

C++

```
CCertificate(IN const CCertificate* pCertificate);
```

Parameters

Parameters	Description
IN const CCertificate* pCertificate	Pointer to the CCertificate to copy.

Description

Copy constructor.

2.12.1.4.2 - Destructors**2.12.1.4.2.1 - CCertificate::~CCertificate Destructor**

Destructor.

C++

```
virtual ~CCertificate();
```

Description

Destructor.

2.12.1.4.3 - Methods**2.12.1.4.3.1 - CCertificate::DisplayCertificate Method**

Displays certificate information.

C++

```
mxt_result DisplayCertificate() const;
```

Returns

resS_OK if there is a valid certificate to display, an error otherwise.

Description

Displays the information regarding the current certificate.

2.12.1.4.3.2 - CCertificate::GetAuthorityKeyIdentifierExtension Method

Gets the Authority Key Identifier extension.

C++

```
mxt_result GetAuthorityKeyIdentifierExtension(OUT CAuthorityKeyIdentifier* pAuthorityKeyIdentifier) const;
```

Parameters

Parameters	Description
OUT CAuthorityKeyIdentifier* pAuthorityKeyIdentifier	Pointer to the outgoing authority key identifier extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the authority key identifier extension.

2.12.1.4.3.3 - CCertificate::GetBasicConstraintsExtension Method

Gets the Basic Constraints extension.

C++

```
mxt_result GetBasicConstraintsExtension(OUT CBasicConstraints* pBasicConstraints) const;
```

Parameters

Parameters	Description
OUT CBasicConstraints* pBasicConstraints	Pointer to the outgoing Basic Constraints extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Basic Constraints extension.

2.12.1.4.3.4 - CCertificate::GetExtendedKeyUsageExtension Method

Gets the Extended Key Usage certificate extension.

C++

```
mxt_result GetExtendedKeyUsageExtension(OUT CExtendedKeyUsage* pExtendedKeyUsage) const;
```

Parameters

Parameters	Description
OUT CExtendedKeyUsage* pExtendedKeyUsage	Pointer to the outgoing Extended Key Usage extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Extended Key Usage extension.

2.12.1.4.3.5 - CCertificate::GetExtension Method

Gets the certificate extension information by index.

C++

```
mxt_result GetExtension(IN unsigned int uExtension, OUT CCertificateExtension* pExtension) const;
```

Parameters

Parameters	Description
IN unsigned int uExtension	Index of the extension to get.
OUT CCertificateExtension* pExtension	Pointer to contain the extension.

Returns

- resFE_INVALID_ARGUMENT
- resS_OK

Description

Gets the extension at the specified index.

2.12.1.4.3.6 - CCertificate::GetExtensionByType Method

Gets the certificate extension corresponding to a specified type.

C++

```
mxt_result GetExtensionByType(IN CCertificateExtension::EType eType, OUT CCertificateExtension* pExtension) const;
```

Parameters

Parameters	Description
IN CCertificateExtension::EType eType	Type of the extension to get.
OUT CCertificateExtension* pExtension	Pointer to contain the extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the certificate extension corresponding to a specified type.

2.12.1.4.3.7 - CCertificate::GetExtensionCount Method

Gets the number of extensions added to this certificate.

C++

```
mxt_result GetExtensionCount(OUT unsigned int* puExtensionCount) const;
```

Parameters

Parameters	Description
OUT unsigned int* puExtensionCount	Pointer to contain the number of extensions.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the number of certificate extensions.

2.12.1.4.3.8 - CCertificate::GetIssuer Method

Gets the certificate issuer.

C++

```
mxt_result GetIssuer(OUT CCertificateIssuer* pIssuer) const;
```

Parameters

Parameters	Description
OUT CCertificateIssuer* pIssuer	Pointer to contain the issuer of this certificate.

Returns

- resFE_INVALID_ARGUMENT
- resS_OK

Description

Gets the issuer of the certificate.

2.12.1.4.3.9 - CCertificate::GetIssuerAlternateNameExtension Method

Gets the Issuer Alternate Name extension.

C++

```
mxt_result GetIssuerAlternateNameExtension(OUT CIssuerAlternateName* pIssuerAlternateName) const;
```

Parameters

Parameters	Description
OUT CIssuerAlternateName* pIssuerAlternateName	Pointer to the outgoing Issuer Alternate Name extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Issuer Alternate Name extension.

2.12.1.4.3.10 - CCertificate::GetKeyUsageExtension Method

Gets the Key Usage certificate extension.

C++

```
mxt_result GetKeyUsageExtension(OUT CKeyUsage* pKeyUsage) const;
```

Parameters

Parameters	Description
OUT CKeyUsage* pKeyUsage	Pointer to the outgoing Key Usage extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Key Usage extension.

2.12.1.4.3.11 - CCertificate::GetNetscapeCertificateTypeExtension Method

Gets the Netscape Certificate Type extension.

C++

```
mxt_result GetNetscapeCertificateTypeExtension(OUT CNetScapeCertificateType* pNetScapeCertificateType) const;
```

Parameters

Parameters	Description
OUT CNetScapeCertificateType* pNetScapeCertificateType	Pointer to the outgoing Netscape Certificate Type extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Netscape Certificate Type extension.

2.12.1.4.3.12 - CCertificate::GetNotAfterTime Method

Gets the time after which the certificate is considered as invalid.

C++

```
mxt_result GetNotAfterTime(OUT CTime* pNotAfter) const;
```

Parameters

Parameters	Description
OUT CTime* pNotAfter	Pointer to contain the time after which a certificate is not valid.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the time after which a certificate is considered invalid.

2.12.1.4.3.13 - CCertificate::GetNotBeforeTime Method

Gets the time before which the certificate is considered as invalid.

C++

```
mxt_result GetNotBeforeTime(OUT CTime* pNotBefore) const;
```

Parameters

Parameters	Description
OUT CTime* pNotBefore	Pointer to contain the time before which a certificate is not valid.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the time before which a certificate is considered invalid.

2.12.1.4.3.14 - CCertificate::GetPrivateKey Method

Gets a private key associated with this X.509 certificate.

C++

```
mxt_result GetPrivateKey(OUT CPrivateKey* pPrivateKey) const;
```

Parameters

Parameters	Description
OUT CPrivateKey* pPrivateKey	Pointer to contain the private key.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the certificate's private key.

NOTES: The key MUST have been set with a call to SetPrivateKey (see page 688) before using this method.

2.12.1.4.3.15 - CCertificate::GetPublicKey Method

Gets the public key associated with this certificate.

C++

```
mxt_result GetPublicKey(OUT CPublicKey* pPublicKey) const;
```

Parameters

Parameters	Description
OUT CPublicKey* pPublicKey	Pointer to contain the public of this certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the public key of the certificate.

2.12.1.4.3.16 - CCertificate::GetSerialNumber Method

Gets the certificate serial number.

C++

```
mxt_result GetSerialNumber(OUT CBlob* pSerialNumber) const;
```

Parameters

Parameters	Description
OUT CBlob* pSerialNumber	Pointer to contain the certificate's serial number.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the certificate's serial number.

2.12.1.4.3.17 - CCertificate::GetSignature Method

Gets the signature and signature algorithm for this certificate.

C++

```
mxt_result GetSignature(OUT CPublicKey::EAlgorithm* peSignatureAlgorithm, OUT CBlob* pSignature) const;
```

Parameters

Parameters	Description
OUT CPublicKey::EAlgorithm* peSignatureAlgorithm	Pointer to contain the signature algorithm.
OUT CBlob* pSignature	Pointer to contain the signature.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the signature of the certificate.

2.12.1.4.3.18 - CCertificate::GetSubject Method

Gets the certificate subject.

C++

```
mxt_result GetSubject(OUT CCertificateSubject* pSubject) const;
```

Parameters

Parameters	Description
OUT CCertificateSubject* pSubject	Pointer to contain the subject of this certificate.

Returns

- resFE_INVALID_ARGUMENT
- resS_OK

Description

Gets the subject of the certificate.

2.12.1.4.3.19 - CCertificate::GetSubjectAlternateNameExtension Method

Gets the Subject Alternate Name extension.

C++

```
mxt_result GetSubjectAlternateNameExtension(OUT CSubjectAlternateName* pSubjectAlternateName) const;
```

Parameters

Parameters	Description
OUT CSubjectAlternateName* pSubjectAlternateName	Pointer to the outgoing Subject Alternate Name extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Subject Alternate Name extension.

2.12.1.4.3.20 - CCertificate::GetSubjectKeyIdentifierExtension Method

Gets the Subject Key Identifier extension.

C++

```
mxt_result GetSubjectKeyIdentifierExtension(OUT CSubjectKeyIdentifier* pSubjectKeyIdentifier) const;
```

Parameters

Parameters	Description
OUT CSubjectKeyIdentifier* pSubjectKeyIdentifier	Pointer to the outgoing subject key identifier extension.

Returns

- resFE_INVALID_ARGUMENT
- resSI_FALSE
- resSI_TRUE

Description

This method gets the Subject key identifier extension.

2.12.1.4.3.21 - CCertificate::GetVersion Method

Gets the certificate version.

C++

```
mxt_result GetVersion(OUT CCertificate::EVersion* peVersion) const;
```

Parameters

Parameters	Description
OUT CCertificate::EVersion* peVersion	Pointer to contain the version of the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the version of the certificate.

2.12.1.4.3.22 - CCertificate::IsIssuedBy Method

Verifies whether a certificate has been issued by another one.

C++

```
mxt_result IsIssuedBy(IN const CCertificate* pCertificate) const;
```

Parameters

Parameters	Description
IN const CCertificate* pCertificate	Pointer that contains the issuing certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resSI_FALSE
- resSI_TRUE

Description

Checks if the current certificate has been issued by the specified certificate. A certificate can verify that it is self-signed by using this method.

2.12.1.4.3.23 - CCertificate::IsSelfIssued Method

Verifies whether a certificate is self-issued.

C++

```
mxt_result IsSelfIssued() const;
```

Returns

- resFE_INVALID_STATE
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the certificates is self-signed.

2.12.1.4.3.24 - CCertificate::Restore Method

Reads a certificate (DER- or PEM-encoded) from a blob.

C++

```
mxt_result Restore(IN const CBlob* pX509);
```

Parameters

Parameters	Description
IN const CBlob* pX509	Pointer containing the certificate

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Restores the certificate from a blob.

2.12.1.4.3.25 - CCertificate::RestoreDer Method

Reads a DER-encoded certificate from a blob.

C++

```
mxt_result RestoreDer(IN const CBlob* pX509);
```

Parameters

Parameters	Description
IN const CBlob* pX509	Pointer to a blob containing the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores to the CCertificate (see page 677) object a certificate stored in DER format in the specified blob.

2.12.1.4.3.26 - CCertificate::RestorePem Method

Reads a PEM-encoded certificate from a blob.

C++

```
mxt_result RestorePem(IN const CBlob* pX509);
```

Parameters

Parameters	Description
IN const CBlob* pX509	Pointer to a blob containing the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores to the CCertificate (see page 677) object a certificate stored in PEM format in the specified blob.

2.12.1.4.3.27 - CCertificate::SetPrivateKey Method

Associates a private key with this X.509 certificate.

C++

```
mxt_result SetPrivateKey(IN const CPrivateKey* pPrivateKey);
```

Parameters

Parameters	Description
IN const CPrivateKey* pPrivateKey	Pointer that contains the private key.

Returns

- resFE_INVALID_ARGUMENT
- resS_OK

Description

Sets the private key associated with this certificate.

2.12.1.4.3.28 - CCertificate::Store Method

Stores the content of this CCertificate (see page 677) into a blob, with a specified encoding.

C++

```
mxt_result Store(OUT CBlob* pX509, IN EEncoding eEncoding = eENCODING_DER) const;
```

Parameters

Parameters	Description
OUT CBlob* pX509	Pointer to contain the certificate.
IN EEncoding eEncoding = eENCODING_DER	Type of encoding used to store the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores the certificate into a blob in the specified encoding.

2.12.1.4.3.29 - CCertificate::StoreDer Method

Stores the content of this certificate in DER format into a blob.

C++

```
mxt_result StoreDer(OUT CBlob* pX509) const;
```

Parameters

Parameters	Description
OUT CBlob* pX509	Pointer to a blob to contain the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores the CCertificate (see page 677) object in DER format in the specified blob.

2.12.1.4.3.30 - CCertificate::StorePem Method

Stores the content of this certificate in PEM format into a blob.

C++

```
mxt_result StorePem(OUT CBlob* pX509) const;
```

Parameters

Parameters	Description
OUT CBlob* pX509	Pointer to a blob to contain the certificate.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Stores the CCertificate (see page 677) object in PEM format in the specified blob.

2.12.1.4.3.31 - VerifySignature**2.12.1.4.3.31.1 - CCertificate::VerifySignature Method**

Validates if the certificate has been properly signed.

C++

```
mxt_result VerifySignature(IN const CCertificate* pIssuerCertificate) const;
```

Parameters

Parameters	Description
IN const CCertificate* pIssuerCertificate	Issuer's certificate.

Returns

resS_OK if successful, res error code otherwise.

Description

Extracts the public key of the issuers certificate to validate the signature of the current certificate.

2.12.1.4.3.31.2 - CCertificate::VerifySignature Method

Validates if the certificate has been properly signed.

C++

```
mxt_result VerifySignature(IN const CPublicKey* pIssuerPublicKey) const;
```

Parameters

Parameters	Description
IN const CPublicKey* pIssuerPublicKey	Issuer's public key.

Returns

resS_OK if successful, res error code otherwise.

Description

Uses the issuer's public key to validate the signature of the current certificate.

2.12.1.4.4 - Operators**2.12.1.4.4.1 - CCertificate::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CCertificate& rCertificate) const;
```

Parameters

Parameters	Description
IN const CCertificate& rCertificate	Reference to the CCertificate (see page 677) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.4.4.2 - CCertificate::= Operator

Assignment operator.

C++

```
CCertificate& operator =(IN const CCertificate& rCertificate);
```

Parameters

Parameters	Description
IN const CCertificate& rCertificate	Reference to the certificate to assign.

Returns

A reference to the assigned certificate.

Description

Assigns the right hand certificate to the left hand one.

2.12.1.4.4.3 - CCertificate::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CCertificate& rCertificate) const;
```

Parameters

Parameters	Description
IN const CCertificate& rCertificate	Reference to the certificate to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.5 - CCertificateChain Class

Class that manages an ordered list of certificates.

Class Hierarchy



C++

```
class CCertificateChain : public MXD_PKI_CCERTIFICATECHAIN_CLASSNAME;
```

Description

CCertificateChain is the class that manages a list of ordered certificates. The certificate chain is transmitted to a remote peer when this peer wants to authenticate the local user/device.

A certificate chain is formed from the following certificates:

- A personal certificate, which identifies the local user or device.
- Zero or more intermediate certificates, which are certificates that identify intermediary Certificate Authorities between the root certificate and the personal certificate.
- Zero or one root certificate, provided by a trusted third-party Certificate Authority (CA).

The ordered list of certificates, starting from the personal certificate and ending at the root CA, usually denotes a delegation of trust where the root CA trusts intermediate CA1, which in turn trusts intermediate CA2, which in turn trusts intermediate CA3, and so on, up to the intermediate CA that has issued the personal certificate to the local user or device, which trusts the user's/device's identity.

Another method is to pass a list of certificates and let the API construct the certificate chain. It orders the certificates with its issuer and places it correctly in the list.

Certificates can be signed with two different signature algorithms: DSA and RSA. The negotiated cipher suite determines whether a DSA or RSA signed personal certificate is needed to identify the local user. CCertificateChain does not allow multiple personal certificates to be added.

It is important that the personal certificate loaded into the certificate chain is configured with its corresponding private key, as this is used to prove (by signing) that this is the real owner of the personal certificate.

The root certificate and intermediate certificates must be properly ordered when they are pushed into the certificate chain with the Extend (see page 693) API. The personal certificate must be inserted first, followed by the issuer of the previous certificates up to the root CA. In other words, the certificates are inserted from the nearest to the farthest, with the personal certificate being inserted first and the root being inserted last.

The loaded intermediate certificates do not have to be configured with a private key, which is usually inaccessible.

Location

Pki/CCertificateChain.h

See Also

CCertificate (see page 677)

Example

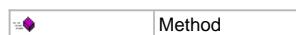
```

Extend(PersonalCa)
Extend(CaN-1DescendantCaN)
...
Extend(Ca1DescendantCa2)
Extend(Ca0DescendantCa1)
Extend(Ca0)
  
```

Constructors

Constructor	Description
CCertificateChain (see page 692)	Constructor.

Legend



Destructors

Destructor	Description
  ~CCertificateChain (see page 693)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 696)	Assignment operator.
 == (see page 696)	Comparison operator.

Legend

	Method
---	--------

Methods

Method	Description
 Clear (see page 693)	Removes all certificates from the certificate chain.
 DisplayCertificateChain (see page 693)	Displays the information for each certificate in the chain.
 Extend (see page 693)	Inserts a certificate within the certificate chain.
 GetCertificates (see page 694)	Gets the list of certificates configured in this chain.
 GetEndEntityCertificate (see page 695)	Gets the certificate of the end entity.
 IsEmpty (see page 695)	Returns whether or not the certificate chain is empty.
 RemoveHighest (see page 695)	Returns and removes the highest PKI (see page 663) hierarchical certificate.
 RemoveLowest (see page 695)	Returns and removes the lowest PKI (see page 663) hierarchical certificate.

Legend

	Method
---	--------

2.12.1.5.1 - Constructors

2.12.1.5.1.1 - CCertificateChain

2.12.1.5.1.1.1 - CCertificateChain::CCertificateChain Constructor

Constructor.

C++

```
CCertificateChain();
```

Description

Basic constructor

2.12.1.5.1.1.2 - CCertificateChain::CCertificateChain Constructor

Copy constructor.

C++

```
CCertificateChain(IN const CCertificateChain& rCertificateChain);
```

Parameters

Parameters	Description
IN const CCertificateChain& rCertificateChain	Reference to a CCertificateChain.

Description

Initializes this certificate chain with the referenced certificate chain.

2.12.1.5.1.1.3 - CCertificateChain::CCertificateChain Constructor

Copy constructor.

C++

```
CCertificateChain(IN const CCertificateChain* pCertificateChain);
```

Parameters

Parameters	Description
IN const CCertificateChain* pCertificateChain	Pointer to a CCertificateChain.

Description

Initializes this certificate chain with the pointed certificate chain.

2.12.1.5.2 - Destructors

2.12.1.5.2.1 - CCertificateChain::~CCertificateChain Destructor

Destructor.

C++

```
virtual ~CCertificateChain();
```

Description

Basic destructor

2.12.1.5.3 - Methods

2.12.1.5.3.1 - CCertificateChain::Clear Method

Removes all certificates from the certificate chain.

C++

```
void Clear();
```

Description

Clears all certificates in the certificate chain.

2.12.1.5.3.2 - CCertificateChain::DisplayCertificateChain Method

Displays the information for each certificate in the chain.

C++

```
mxt_result DisplayCertificateChain();
```

Returns

resS_OK if there is a non-empty certificate chain to display and that all certificates are valid, an error otherwise.

Description

Displays the information of the certificate chain. The chain is displayed from the lowest PKI (see page 663) hierarchical certificate to the highest.

2.12.1.5.3.3 - Extend

2.12.1.5.3.3.1 - CCertificateChain::Extend Method

Inserts a certificate within the certificate chain.

C++

```
mxt_result Extend(IN const CCertificate* pCertificate, IN bool bInsertRootCertificate = true);
```

Parameters

Parameters	Description
IN const CCertificate* pCertificate	Pointer to the certificate to append to the chain.
IN bool bInsertRootCertificate = true	bool to determine whether or not to insert a root certificate into the chain.

Returns

resS_OK if the certificate has been successfully added, it is already in the chain, or it is a root certificate pertaining to the current chain and the bInsertRootCertificate flag is false. If the flag is true, the root certificate is added as any other certificate. Returns mxt_result (see page 92) error code otherwise.

Description

Inserts a certificate at the end of the certificate chain. Only a higher PKI (see page 663) level certificate can be added with this method.

A valid certificate is always appended to an empty chain. If a chain is already present, the certificate is added only if it issued the previous certificate.

2.12.1.5.3.3.2 - CCertificateChain::Extend Method

Adds new certificates to the certificate chain.

C++

```
mxt_result Extend(IN const CVector<CCertificate>* pvecCertificates, IN bool bInsertRootCertificate = true, IN bool bEnforceRelation = true);
```

Parameters

Parameters	Description
IN const CVector<CCertificate>* pvecCertificates	Pointer to a vector containing the certificates to add.
IN bool bInsertRootCertificate = true	bool to determine whether or not to insert the root certificates into the chain.
IN bool bEnforceRelation = true	bool flag to determine how to handle possible multiple chains.

Returns

resS_OK if successful, mxt_result (see page 92) error code otherwise.

Description

Adds any certificate in the vector to the current certificate chain.

Any root certificate is ignored if the bInsertRootCertificate is set to false.

If the chain is empty, it builds the chain from the list. If there are multiple initial chains, then the method fails and no certificate chain is created.

In the case of an already existing chain, the bEnforceRelation determines the handling of multiple chains. If the flag is set to false, it extracts the items to append to the certificate chain and adds them to the current chain. If it is set to true, it extends the chain only if all certificates in the vector descend from the last certificate currently in the chain. Certificates already in the current chain are ignored. In both cases, if a lower PKI (see page 663) level is in the vector, the function fails and the chain remains unchanged.

Any invalid certificate causes an error.

2.12.1.5.3.4 - CCertificateChain::GetCertificates Method

Gets the list of certificates configured in this chain.

C++

```
mxt_result GetCertificates(OUT CVector<CCertificate>* pvecCertificates) const;
```

Parameters

Parameters	Description
OUT CVector<CCertificate>* pvecCertificates	Vector to hold the certificate chain.

Returns

resS_OK if successful, resFE_INVALID_ARGUMENT otherwise.

Description

Gets the list of certificates configured in this chain. The returned vector contains the ordered certificate list.

2.12.1.5.3.5 - CCertificateChain::GetEndEntityCertificate Method

Gets the certificate of the end entity.

C++

```
mxt_result GetEndEntityCertificate(OUT CCertificate* pCertificate) const;
```

Parameters

Parameters	Description
OUT CCertificate* pCertificate	Pointer to the certificate.

Returns

resS_OK resFE_INVALID_ARGUMENT if the pointer is NULL resSI_FALSE if there is no end entity certificate

Description

Gets the certificate of the end entity. The end entity is the first certificate in the chain, the one that is verified by all the other certificates in the chain.

2.12.1.5.3.6 - CCertificateChain::IsEmpty Method

Returns whether or not the certificate chain is empty.

C++

```
bool IsEmpty();
```

Returns

True if the chain is empty, false otherwise.

Description

Returns whether or not the certificate chain is empty.

2.12.1.5.3.7 - CCertificateChain::RemoveHighest Method

Returns and removes the highest PKI (see page 663) hierarchical certificate.

C++

```
mxt_result RemoveHighest(OUT CCertificate* pCertificate);
```

Parameters

Parameters	Description
OUT CCertificate* pCertificate	Pointer to hold the removed certificate.

Returns

resS_OK if successful, resFE_INVALID_STATE otherwise.

Description

Removes the highest PKI (see page 663) hierarchical certificate in the list. If the pCertificate pointer is NULL, then the certificate is simply discarded; otherwise, it is copied before being removed.

2.12.1.5.3.8 - CCertificateChain::RemoveLowest Method

Returns and removes the lowest PKI (see page 663) hierarchical certificate.

C++

```
mxt_result RemoveLowest(OUT CCertificate* pCertificate);
```

Parameters

Parameters	Description
OUT CCertificate* pCertificate	Pointer to hold the removed certificate.

Returns

resS_OK if successful, resFE_INVALID_STATE otherwise.

Description

Removes the lowest PKI (see page 663) hierarchical certificate in the list. If the pCertificate pointer is NULL, then the certificate is simply discarded; otherwise, it is copied before being removed.

2.12.1.5.4 - Operators

2.12.1.5.4.1 - CCertificateChain::= Operator

Assignment operator.

C++

```
CCertificateChain& operator =(IN const CCertificateChain& rCertificateChain);
```

Parameters

Parameters	Description
IN const CCertificateChain& rCertificateChain	Referenced certificate chain.

Returns

Reference to the current CCertificateChain (see page 691).

Description

Sets the current certificate chain equal to the referenced one. This is not a deep copy of the referenced chain.

2.12.1.5.4.2 - CCertificateChain::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CCertificateChain& rCertificateChain) const;
```

Parameters

Parameters	Description
IN const CCertificateChain& rCertificateChain	Referenced certificate chain.

Returns

true if both equal, false otherwise.

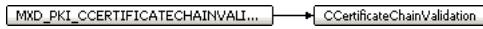
Description

Compares the internal vectors of both certificate chains to see if they are equal.

2.12.1.6 - CCertificateChainValidation Class

Performs certificate chain validation.

Class Hierarchy



C++

```
class CCertificateChainValidation : public MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME;
```

Description

CCertificateChainValidation is the class that validates a chain of certificates.

The validation process checks the integrity and authenticity of all certificates within a certificate chain against a vector of trusted and

untrusted certificates. The process may take a long time, depending on the length of the chain, the algorithms used to sign the certificates, and the length of their keys.

Location

Pki/CCertificateChainValidation.h

See Also

CCertificate (see page 677), CCertificateChain (see page 691), CTime (see page 822)

Constructors

Constructor	Description
CCertificateChainValidation (see page 697)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CCertificateChainValidation (see page 697)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
SetTrustedCertificates (see page 698)	Sets the trusted certificates.
SetUntrustedCertificates (see page 698)	Sets the untrusted certificates.
Validate (see page 698)	Performs the certificate chain's validation.

Legend

	Method
---	--------

2.12.1.6.1 - Constructors

2.12.1.6.1.1 - CCertificateChainValidation::CCertificateChainValidation Constructor

Constructor.

C++

```
CCertificateChainValidation();
```

Description

Default constructor.

2.12.1.6.2 - Destructors

2.12.1.6.2.1 - CCertificateChainValidation::~CCertificateChainValidation Destructor

Destructor.

C++

```
virtual ~CCertificateChainValidation();
```

Description

Basic destructor

2.12.1.6.3 - Methods

2.12.1.6.3.1 - CCertificateChainValidation::SetTrustedCertificates Method

Sets the trusted certificates.

C++

```
mxt_result SetTrustedCertificates(IN const CVector<CCertificate>* pvecTrustedCertificates);
```

Parameters

Parameters	Description
IN const CVector<CCertificate>* pvecTrustedCertificates	Pointer to the vector of trusted certificates.

Returns

resS_OK if successful, mxt_result (see page 92) error code otherwise.

Description

Sets the vector of trusted certificates. It is used if no trusted certificate is provided to the Validate (see page 698) method.

2.12.1.6.3.2 - CCertificateChainValidation::SetUntrustedCertificates Method

Sets the untrusted certificates.

C++

```
mxt_result SetUntrustedCertificates(IN const CVector<CCertificate>* pvecUntrustedCertificates);
```

Parameters

Parameters	Description
IN const CVector<CCertificate>* pvecUntrustedCertificates	Pointer to the vector of untrusted certificates.

Returns

resS_OK if successful, mxt_result (see page 92) error code otherwise.

Description

Sets the vector of untrusted certificates. It is used if no untrusted certificate is provided to the Validate (see page 698) method.

2.12.1.6.3.3 - CCertificateChainValidation::Validate Method

Performs the certificate chain's validation.

C++

```
mxt_result Validate(IN const CCertificateChain* pCertificateChain, IN const CVector<CCertificate>* pvecTrustedCertificates, IN const CVector<CCertificate>* pvecUntrustedCertificates, IN const CTime* pTime);
```

Parameters

Parameters	Description
IN const CCertificateChain* pCertificateChain	Pointer to the certificate chain.
IN const CVector<CCertificate>* pvecTrustedCertificates	Pointer to the Trusted certificates. If NULL is passed, the internal trusted certificates are used instead.
IN const CVector<CCertificate>* pvecUntrustedCertificates	Pointer to the untrusted certificates. If NULL, the internal untrusted certificates are used instead.
IN const CTime* pTime	Pointer to the time against which the Certificates Time validity period is verified. If NULL is passed, the system time is used instead.

Returns

resSI_FALSE: Certificate chain is invalid.

resSI_TRUE: Certificate chain is valid.

resFE_INVALID_ARGUMENT: Invalid argument.

Description

This method performs the certificate chain's validation.

2.12.1.7 - CCertificateExtension Class

Class managing certificate extensions.

Class Hierarchy



C++

```
class CCertificateExtension : public MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME;
```

Description

CCertificateExtension is the class that manages the certificate extensions.

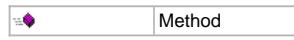
Location

Pki/CCertificateExtension.h

Constructors

Constructor	Description
CCertificateExtension (see page 700)	Constructor.

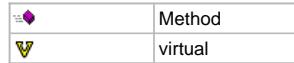
Legend



Destructors

Destructor	Description
~CCertificateExtension (see page 700)	Destructor.

Legend



Operators

Operator	Description
!= (see page 703)	Different than operator.
= (see page 703)	Assignment operator.
== (see page 704)	Comparison operator.

Legend



Methods

Method	Description
GetBasicConstraints (see page 700)	Gets the Basic Constraints extension.
GetExtendedKeyUsage (see page 701)	Gets the Extended key usage.
GetIssuerAlternateName (see page 701)	Gets the Issuer Alternate Name extension.
GetKeyUsage (see page 701)	Gets the Key usage extension.
GetNetscapeCertificateType (see page 702)	Gets the Netscape certificate extensions.
GetSubjectAlternateName (see page 702)	Gets the Subject Alternate Name extension.
GetType (see page 702)	Gets the type of extension.
IsCritical (see page 703)	Verifies if an extension is critical.

Legend



2.12.1.7.1 - Constructors

2.12.1.7.1.1 - CCertificateExtension

2.12.1.7.1.1.1 - CCertificateExtension::CCertificateExtension Constructor

Constructor.

C++

```
CCertificateExtension();
```

Description

Constructor.

2.12.1.7.1.1.2 - CCertificateExtension::CCertificateExtension Constructor

Copy constructor.

C++

```
CCertificateExtension(IN const CCertificateExtension& rCertificateExtension);
```

Parameters

Parameters	Description
IN const CCertificateExtension& rCertificateExtension	Reference to the CCertificateExtension to copy.

Description

Copy constructor.

2.12.1.7.1.1.3 - CCertificateExtension::CCertificateExtension Constructor

Copy constructor.

C++

```
CCertificateExtension(IN const CCertificateExtension* pCertificateExtension);
```

Parameters

Parameters	Description
IN const CCertificateExtension* pCertificateExtension	Pointer to the CCertificateExtension to copy.

Description

Copy constructor.

2.12.1.7.2 - Destructors

2.12.1.7.2.1 - CCertificateExtension::~CCertificateExtension Destructor

Destructor.

C++

```
virtual ~CCertificateExtension();
```

Description

Destructor.

2.12.1.7.3 - Methods

2.12.1.7.3.1 - CCertificateExtension::GetBasicConstraints Method

Gets the Basic Constraints extension.

C++

```
mxt_result GetBasicConstraints(OUT CBasicConstraints* pBasicConstraints) const;
```

Parameters

Parameters	Description
OUT CBasicConstraints* pBasicConstraints	Pointer to contain the Basic Constraints extension.

Returns

resFE_FAIL resS_OK.

Description

<< Accessors >>

This method gets the Basic Constraints extension.

2.12.1.7.3.2 - CCertificateExtension::GetExtendedKeyUsage Method

Gets the Extended key usage.

C++

```
mxt_result GetExtendedKeyUsage(OUT CExtendedKeyUsage* pExtendedKeyUsage) const;
```

Parameters

Parameters	Description
OUT CExtendedKeyUsage* pExtendedKeyUsage	Pointer to contain the Extended Key Usage extension.

Returns

resFE_FAIL resS_OK.

Description

This method gets the Extended Key Usage extension.

2.12.1.7.3.3 - CCertificateExtension::GetIssuerAlternateName Method

Gets the Issuer Alternate Name extension.

C++

```
mxt_result GetIssuerAlternateName(OUT CIssuerAlternateName* pIssuerAlternateName) const;
```

Parameters

Parameters	Description
OUT CIssuerAlternateName* pIssuerAlternateName	Pointer to contain the Issuer Alternate Name extension.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

This method gets the Issuer Alternate Name extension.

2.12.1.7.3.4 - CCertificateExtension::GetKeyUsage Method

Gets the Key usage extension.

C++

```
mxt_result GetKeyUsage(OUT CKeyUsage* pKeyUsage) const;
```

Parameters

Parameters	Description
OUT CKeyUsage* pKeyUsage	Pointer to contain the Key Usage extension.

Returns

resFE_FAIL resS_OK.

Description

This method gets the Key Usage extension.

2.12.1.7.3.5 - CCertificateExtension::GetNetscapeCertificateType Method

Gets the Netscape certificate extensions.

C++

```
mxt_result GetNetscapeCertificateType(OUT CNetscapeCertificateType* pNetscapeCertificateType) const;
```

Parameters

Parameters	Description
OUT CNetscapeCertificateType* pNetscapeCertificateType	Pointer to contain the Netscape Certificate Type extension.

Returns

resFE_FAIL resS_OK.

Description

This method gets the Netscape Certificate Type extension.

2.12.1.7.3.6 - CCertificateExtension::GetSubjectAlternateName Method

Gets the Subject Alternate Name extension.

C++

```
mxt_result GetSubjectAlternateName(OUT CSubjectAlternateName* pSubjectAlternateName) const;
```

Parameters

Parameters	Description
OUT CSubjectAlternateName* pSubjectAlternateName	Pointer to contain the Subject Alternate Name extension.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

This method gets the Subject Alternate Name extension.

2.12.1.7.3.7 - CCertificateExtension::GetType Method

Gets the type of extension.

C++

```
mxt_result GetType(OUT EType* peType) const;
```

Parameters

Parameters	Description
OUT EType* peType	Pointer to contain the extension type.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Gets the type of certificate extension.

2.12.1.7.3.8 - CCertificateExtension::IsCritical Method

Verifies if an extension is critical.

C++

```
mxt_result IsCritical() const;
```

Returns

- resFE_FAIL
- resFE_INVALID_STATE
- resSI_FALSE
- resSI_TRUE

Description

Verifies if an extension is critical.

NOTES: Unrecognized extensions may be ignored if IsCritical() returns resSI_FALSE. If IsCritical() returns resSI_TRUE, then the extension is considered invalid and the certificate must also be considered invalid.

2.12.1.7.4 - Operators**2.12.1.7.4.1 - CCertificateExtension::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CCertificateExtension& rCertificateExtension) const;
```

Parameters

Parameters	Description
IN const CCertificateExtension& rCertificateExtension	Reference to the CCertificateExtension (see page 699) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.7.4.2 - CCertificateExtension::= Operator

Assignment operator.

C++

```
CCertificateExtension& operator =(IN const CCertificateExtension& rCertificateExtension);
```

Parameters

Parameters	Description
IN const CCertificateExtension& rCertificateExtension	Reference to the CCertificateExtension (see page 699) to assign.

Returns

A reference to the assigned extension.

Description

Assigns the right hand object to the left hand one.

2.12.1.7.4.3 - CCertificateExtension::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CCertificateExtension& rCertificateExtension) const;
```

Parameters

Parameters	Description
IN const CCertificateExtension& rCertificateExtension	Reference to the CCertificateExtension (see page 699) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.8 - CCertificateIssuer Class

Class managing the certificate issuer related identifiers and values.

Class Hierarchy



C++

```
class CCertificateIssuer : public MXD_PKI_CCERTIFICATEISSUER_CLASSNAME;
```

Description

CCertificateIssuer is the class that manages the certificate issuer related identifiers and values.

Location

Pki/CCertificateIssuer.h

Constructors

Constructor	Description
CCertificateIssuer (see page 705)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CCertificateIssuer (see page 705)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 706)	Different than operator.
= (see page 707)	Assignment operator.
== (see page 707)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
GetName (see page 706)	Gets the specified name.
GetNames (see page 706)	Gets the vector of all names.

Legend

	Method
--	--------

2.12.1.8.1 - Constructors**2.12.1.8.1.1 - CCertificateIssuer****2.12.1.8.1.1.1 - CCertificateIssuer::CCertificateIssuer Constructor**

Constructor.

C++

```
CCertificateIssuer();
```

Description

Constructor.

2.12.1.8.1.1.2 - CCertificateIssuer::CCertificateIssuer Constructor

Copy constructor.

C++

```
CCertificateIssuer(IN const CCertificateIssuer& rCertificateIssuer);
```

Parameters

Parameters	Description
IN const CCertificateIssuer& rCertificateIssuer	Reference to the CCertificateIssuer to compare.

Description

Copy constructor.

2.12.1.8.1.1.3 - CCertificateIssuer::CCertificateIssuer Constructor

Copy constructor.

C++

```
CCertificateIssuer(IN const CCertificateIssuer* pCertificateIssuer);
```

Parameters

Parameters	Description
IN const CCertificateIssuer* pCertificateIssuer	Pointer to the CCertificateIssuer to compare.

Description

Copy constructor.

2.12.1.8.2 - Destructors**2.12.1.8.2.1 - CCertificateIssuer::~CCertificateIssuer Destructor**

Destructor.

C++

```
virtual ~CCertificateIssuer();
```

Description

Destructor.

2.12.1.8.3 - Methods

2.12.1.8.3.1 - CCertificateIssuer::GetName Method

Gets the specified name.

C++

```
mxt_result GetName(IN EName eName, OUT CString* pstrName, OUT bool* pbExist) const;
```

Parameters

Parameters	Description
IN EName eName	Name to get in the issuer.
OUT CString* pstrName	Pointer to contain the string of the name.
OUT bool* pbExist	Pointer to contain true if the requested name exists, false otherwise.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Gets the specified name from the issuer.

2.12.1.8.3.2 - CCertificateIssuer::GetNames Method

Gets the vector of all names.

C++

```
mxt_result GetNames(IN EName eName, OUT CVector<CString>* pvecstrNames) const;
```

Parameters

Parameters	Description
IN EName eName	Name to get in the issuer.
OUT CVector<CString>* pvecstrNames	Pointer to a vector containing the names. MUST not be NULL.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Gets all the names corresponding to the EName from the issuer.

2.12.1.8.4 - Operators

2.12.1.8.4.1 - CCertificateIssuer::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CCertificateIssuer& rCertificateIssuer) const;
```

Parameters

Parameters	Description
IN const CCertificateIssuer& rCertificateIssuer	Reference to the CCertificateIssuer (see page 704) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.8.4.2 - CCertificateIssuer::= Operator

Assignment operator.

C++

```
CCertificateIssuer& operator =(IN const CCertificateIssuer& rCertificateIssuer);
```

Parameters

Parameters	Description
IN const CCertificateIssuer& rCertificateIssuer	Reference to the CCertificateIssuer (see page 704) to assign.

Returns

A reference to the assigned issuer.

Description

Assigns the right hand issuer to the left hand one.

2.12.1.8.4.3 - CCertificateIssuer::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CCertificateIssuer& rCertificateIssuer) const;
```

Parameters

Parameters	Description
IN const CCertificateIssuer& rCertificateIssuer	Reference to the CCertificateIssuer (see page 704) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.9 - CCertificateSubject Class

Class that manages the certificate subject related identifiers and values.

Class Hierarchy**C++**

```
class CCertificateSubject : public MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME;
```

Description

CCertificateSubject is the class that manages the certificate subject related identifiers and values.

Location

Pki/CCertificateSubject.h

Constructors

Constructor	Description
CCertificateSubject (see page 708)	Constructor.

Legend

Destructors

Destructor	Description
  ~CCertificateSubject (see page 709)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 710)	Different than operator.
 = (see page 710)	Assignment operator.
 == (see page 710)	Comparison operator.

Legend

	Method
---	--------

Methods

Method	Description
 GetName (see page 709)	Gets the specified name.
 GetNames (see page 709)	Gets the vector of all names.

Legend

	Method
---	--------

2.12.1.9.1 - Constructors

2.12.1.9.1.1 - CCertificateSubject

2.12.1.9.1.1.1 - CCertificateSubject::CCertificateSubject Constructor

Constructor.

C++

```
CCertificateSubject();
```

Description

Constructor.

2.12.1.9.1.1.2 - CCertificateSubject::CCertificateSubject Constructor

Copy constructor.

C++

```
CCertificateSubject(IN const CCertificateSubject& rCertificateSubject);
```

Parameters

Parameters	Description
IN const CCertificateSubject& rCertificateSubject	Reference to the CCertificateSubject to compare.

Description

Copy constructor.

2.12.1.9.1.1.3 - CCertificateSubject::CCertificateSubject Constructor

Copy constructor.

C++

```
CCertificateSubject(IN const CCertificateSubject* pCertificateSubject);
```

Parameters

Parameters	Description
IN const CCertificateSubject* pCertificateSubject	Pointer to the CCertificateSubject to compare.

Description

Copy constructor.

2.12.1.9.2 - Destructors**2.12.1.9.2.1 - CCertificateSubject::~CCertificateSubject Destructor**

Destructor.

C++

```
virtual ~CCertificateSubject();
```

Description

Destructor.

2.12.1.9.3 - Methods**2.12.1.9.3.1 - CCertificateSubject::GetName Method**

Gets the specified name.

C++

```
mxt_result GetName(IN EName eName, OUT CString* pstrName, OUT bool* pbExist) const;
```

Parameters

Parameters	Description
IN EName eName	Name to get in the subject
OUT CString* pstrName	Pointer to contain the string of the name.
OUT bool* pbExist	Pointer to contain true if the requested name exists, false otherwise.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Gets the specified name from the subject.

2.12.1.9.3.2 - CCertificateSubject::GetNames Method

Gets the vector of all names.

C++

```
mxt_result GetNames(IN EName eName, OUT CVector<CString>* pvecstrNames) const;
```

Parameters

Parameters	Description
IN EName eName	Name to get in the subject
OUT CVector<CString>* pvecstrNames	Pointer to a vector containing the names. MUST not be NULL.

Returns

- resFE_INVALID_ARGUMENT

- resFE_INVALID_STATE
- resFE_FAIL
- resS_OK

Description

Gets all the names corresponding to the EName from the subject.

2.12.1.9.4 - Operators

2.12.1.9.4.1 - CCertificateSubject::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CCertificateSubject& rCertificateSubject) const;
```

Parameters

Parameters	Description
IN const CCertificateSubject& rCertificateSubject	Reference to the CCertificateSubject (see page 707) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.9.4.2 - CCertificateSubject::= Operator

Assignment operator.

C++

```
CCertificateSubject& operator =(IN const CCertificateSubject& rCertificateSubject);
```

Parameters

Parameters	Description
IN const CCertificateSubject& rCertificateSubject	Reference to the CCertificateSubject (see page 707) to assign.

Returns

A reference to the assigned subject.

Description

Assigns the right hand subject to the left hand one.

2.12.1.9.4.3 - CCertificateSubject::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CCertificateSubject& rCertificateSubject) const;
```

Parameters

Parameters	Description
IN const CCertificateSubject& rCertificateSubject	Reference to the CCertificateSubject (see page 707) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.10 - CExtendedKeyUsage Class

Class to handle the extended key usage extensions.

Class Hierarchy



C++

```
class CExtendedKeyUsage : public MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME;
```

Description

CExtendedKeyUsage is the class to handle the extended key usage extensions inside certificates.

Location

Pki/CExtendedKeyUsage.h

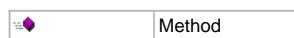
See Also

CCertificateExtension (see page 699)

Constructors

Constructor	Description
~CExtendedKeyUsage (see page 712)	Constructor.

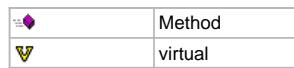
Legend



Destructors

Destructor	Description
~CExtendedKeyUsage (see page 712)	Destructor.

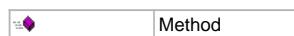
Legend



Operators

Operator	Description
!= (see page 714)	Different than operator.
== (see page 714)	Assignment operator.
== (see page 715)	Comparison operator.

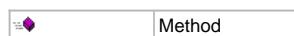
Legend



Methods

Method	Description
IsClientAuthenticationSet (see page 712)	Verifies if the client authentication flag is set.
IsCodeSignedSet (see page 713)	Verifies if the code signed flag is set.
IsOcspSignedSet (see page 713)	Verifies if the OSCP signed flag is set.
IsServerAuthenticationSet (see page 713)	Verifies if the server authentication flag is set.
IsSmimeSet (see page 713)	Verifies if the SMIME flag is set.
IsTimestampingSet (see page 714)	Verifies if the timestamping flag is set.

Legend



2.12.1.10.1 - Constructors

2.12.1.10.1.1 - CExtendedKeyUsage

2.12.1.10.1.1.1 - CExtendedKeyUsage::CExtendedKeyUsage Constructor

Constructor.

C++

```
CExtendedKeyUsage();
```

Description

Constructor.

2.12.1.10.1.1.2 - CExtendedKeyUsage::CExtendedKeyUsage Constructor

Copy constructor.

C++

```
CExtendedKeyUsage(IN const CExtendedKeyUsage& rExtendedKeyUsage);
```

Parameters

Parameters	Description
IN const CExtendedKeyUsage& rExtendedKeyUsage	Reference to the object to copy.

Description

Constructor.

2.12.1.10.1.1.3 - CExtendedKeyUsage::CExtendedKeyUsage Constructor

Copy constructor.

C++

```
CExtendedKeyUsage(IN const CExtendedKeyUsage* pExtendedKeyUsage);
```

Parameters

Parameters	Description
IN const CExtendedKeyUsage* pExtendedKeyUsage	Pointer to the object to copy.

Description

Constructor.

2.12.1.10.2 - Destructors

2.12.1.10.2.1 - CExtendedKeyUsage::~CExtendedKeyUsage Destructor

Destructor.

C++

```
virtual ~CExtendedKeyUsage();
```

Description

Destructor.

2.12.1.10.3 - Methods

2.12.1.10.3.1 - CExtendedKeyUsage::IsClientAuthenticationSet Method

Verifies if the client authentication flag is set.

C++

```
mxt_result IsClientAuthenticationSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the client authentication flag is set.

2.12.1.10.3.2 - CExtendedKeyUsage::IsCodeSignedSet Method

Verifies if the code signed flag is set.

C++

```
mxt_result IsCodeSignedSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the code signed flag is set.

2.12.1.10.3.3 - CExtendedKeyUsage::IsOcspSignedSet Method

Verifies if the OSCP signed flag is set.

C++

```
mxt_result IsOcspSignedSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the OSCP signed flag is set.

2.12.1.10.3.4 - CExtendedKeyUsage::IsServerAuthenticationSet Method

Verifies if the server authentication flag is set.

C++

```
mxt_result IsServerAuthenticationSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the server authentication flag is set.

2.12.1.10.3.5 - CExtendedKeyUsage::IsSmimeSet Method

Verifies if the SMIME flag is set.

C++

```
mxt_result IsSmimeSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the SMIME flag is set.

2.12.1.10.3.6 - CExtendedKeyUsage::IsTimestampingSet Method

Verifies if the timestamping flag is set.

C++

```
mxt_result IsTimestampingSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the timestamping flag is set.

2.12.1.10.4 - Operators**2.12.1.10.4.1 - CExtendedKeyUsage::!= Operator**

Different than operator.

C++

```
bool operator !=(IN const CExtendedKeyUsage& rExtendedKeyUsage) const;
```

Parameters

Parameters	Description
IN const CExtendedKeyUsage& rExtendedKeyUsage	Reference to the CExtendedKeyUsage (see page 711) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.10.4.2 - CExtendedKeyUsage::= Operator

Assignment operator.

C++

```
CExtendedKeyUsage& operator =(IN const CExtendedKeyUsage& rExtendedKeyUsage);
```

Parameters

Parameters	Description
IN const CExtendedKeyUsage& rExtendedKeyUsage	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.10.4.3 - CExtendedKeyUsage::== Operator

Comparison operator.

C++

```
bool operator ==( IN const CExtendedKeyUsage& rExtendedKeyUsage) const;
```

Parameters

Parameters	Description
IN const CExtendedKeyUsage& rExtendedKeyUsage	Reference to the CExtendedKeyUsage (see page 711) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.11 - CIssuerAlternateName Class

Class that manages the issuerAlternateName extension.

Class Hierarchy**C++**

```
class CIssuerAlternateName : public CAAlternateName;
```

Description

CIssuerAlternateName is the class that manages the issuerAlternateName extension.

Location

Pki/CIssuerAlternateName.h

Constructors

Constructor	Description
• CIssuerAlternateName (see page 716)	Constructor.

CAAlternateName Class

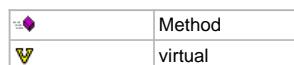
CAAlternateName Class	Description
• CAAlternateName (see page 665)	Constructor.

Legend**Destructors**

Destructor	Description
• V ~CIssuerAlternateName (see page 717)	Destructor.

CAAlternateName Class

CAAlternateName Class	Description
• V ~CAAlternateName (see page 665)	Destructor.

Legend

Operators

Operator	Description
!= (see page 717)	Different than operator.
= (see page 717)	Assignment operator.
== (see page 718)	Comparison operator.

CAAlternateName Class

CAAlternateName Class	Description
!= (see page 667)	Different than operator.
= (see page 668)	Assignment operator.
== (see page 668)	Comparison operator.

Legend

	Method
--	--------

Methods

CAAlternateName Class

CAAlternateName Class	Description
GetDnsName (see page 665)	Gets the value of a name of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.
GetIpAddress (see page 666)	Gets the value of a name of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.
GetNameCount (see page 666)	Gets the number of name entries.
GetNameType (see page 666)	Gets the type of a name entry.
GetRfc822Name (see page 667)	Gets the value of the name of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.
GetUniformResourceIdentifier (see page 667)	Gets the value of a name of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

Legend

	Method
--	--------

2.12.1.11.1 - Constructors

2.12.1.11.1.1 - CIssuerAlternateName

2.12.1.11.1.1.1 - CIssuerAlternateName::CIssuerAlternateName Constructor

Constructor.

C++

```
CIssuerAlternateName();
```

Description

Constructor.

2.12.1.11.1.1.2 - CIssuerAlternateName::CIssuerAlternateName Constructor

Copy constructor.

C++

```
CIssuerAlternateName( IN const CIssuerAlternateName& rIssuerAlternateName );
```

Parameters

Parameters	Description
IN const CIssuerAlternateName& rIssuerAlternateName	Reference to the CIssuerAlternateName to copy.

Description

Copy constructor.

2.12.1.11.1.1.3 - CIssuerAlternateName::CIssuerAlternateName Constructor

Copy constructor.

C++

```
CIssuerAlternateName( IN const CIssuerAlternateName* pIssuerAlternateName );
```

Parameters

Parameters	Description
IN const CIssuerAlternateName* pIssuerAlternateName	Pointer to the CIssuerAlternateName to copy.

Description

Copy constructor.

2.12.1.11.2 - Destructors

2.12.1.11.2.1 - CIssuerAlternateName::~CIssuerAlternateName Destructor

Destructor.

C++

```
virtual ~CIssuerAlternateName();
```

Description

Destructor.

2.12.1.11.3 - Operators

2.12.1.11.3.1 - CIssuerAlternateName::!= Operator

Different than operator.

C++

```
bool operator !=( IN const CIssuerAlternateName& rIssuerAlternateName ) const;
```

Parameters

Parameters	Description
IN const CIssuerAlternateName& rIssuerAlternateName	Reference to the CIssuerAlternateName (see page 715) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.11.3.2 - CIssuerAlternateName:::= Operator

Assignment operator.

C++

```
CIssuerAlternateName& operator =( IN const CIssuerAlternateName& rIssuerAlternateName );
```

Parameters

Parameters	Description
IN const CIssuerAlternateName& rIssuerAlternateName	Reference to the CIssuerAlternateName (see page 715) to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.11.3.3 - CIssuerAlternateName::== Operator

Comparison operator.

C++

```
bool operator ==( IN const CIssuerAlternateName& rIssuerAlternateName) const;
```

Parameters

Parameters	Description
IN const CIssuerAlternateName& rIssuerAlternateName	Reference to the CIssuerAlternateName (see page 715) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.12 - CKeyUsage Class

Class to handle the key usage extensions inside certificates.

Class Hierarchy**C++**

```
class CKeyUsage : public MXD_PKI_CKEYUSAGE_CLASSNAME;
```

Description

CKeyUsage is the class to handle the key usage extensions inside certificates.

Location

Pki/CKeyUsage.h

See Also

CCertificateExtension (see page 699)

Constructors

Constructor	Description
~CKeyUsage (see page 719)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CKeyUsage (see page 720)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 722)	Different than operator.
= (see page 722)	Assignment operator.
== (see page 723)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
• IsCertificateSigningSet (see page 720)	Verifies if the certificate signing flag is set.
• IsCrlSignedSet (see page 720)	Verifies if the CRL signed flag is set.
• IsDataEnciphermentSet (see page 720)	Verifies if the data encipherment flag is set.
• IsDecipherOnlySet (see page 721)	Verifies if the decipher only flag is set.
• IsDigitalSignatureSet (see page 721)	Verifies if the digital signature flag is set.
• IsEncipherOnlySet (see page 721)	Verifies if the encipher only flag is set.
• IsKeyAgreementSet (see page 721)	Verifies if the key agreement flag is set.
• IsKeyEnciphermentSet (see page 722)	Verifies if the key encipherment flag is set.
• IsNonRepudiationSet (see page 722)	Verifies if the non repudiation flag is set.

Legend

	Method
--	--------

2.12.1.12.1 - Constructors

2.12.1.12.1.1 - CKeyUsage

2.12.1.12.1.1.1 - CKeyUsage::CKeyUsage Constructor

Constructor.

C++

```
CKeyUsage();
```

Description

Constructor.

2.12.1.12.1.1.2 - CKeyUsage::CKeyUsage Constructor

Copy constructor.

C++

```
CKeyUsage( IN const CKeyUsage& rKeyUsage );
```

Parameters

Parameters	Description
IN const CKeyUsage& rKeyUsage	Reference to the object to copy.

Description

Constructor.

2.12.1.12.1.1.3 - CKeyUsage::CKeyUsage Constructor

Copy constructor.

C++

```
CKeyUsage( IN const CKeyUsage* pKeyUsage );
```

Parameters

Parameters	Description
IN const CKeyUsage* pKeyUsage	Pointer to the object to copy.

Description

Constructor.

2.12.1.12.2 - Destructors

2.12.1.12.2.1 - CKeyUsage::~CKeyUsage Destructor

Destructor.

C++

```
virtual ~CKeyUsage();
```

Description

Destructor.

2.12.1.12.3 - Methods

2.12.1.12.3.1 - CKeyUsage::IsCertificateSigningSet Method

Verifies if the certificate signing flag is set.

C++

```
mxt_result IsCertificateSigningSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the certificate signing flag is set.

2.12.1.12.3.2 - CKeyUsage::IsCrlSignedSet Method

Verifies if the CRL signed flag is set.

C++

```
mxt_result IsCrlSignedSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the CRL signed flag is set.

2.12.1.12.3.3 - CKeyUsage::IsDataEnciphermentSet Method

Verifies if the data encipherment flag is set.

C++

```
mxt_result IsDataEnciphermentSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the data encipherment flag is set.

2.12.1.12.3.4 - CKeyUsage::IsDecipherOnlySet Method

Verifies if the decipher only flag is set.

C++

```
mxt_result IsDecipherOnlySet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the decipher only flag is set.

2.12.1.12.3.5 - CKeyUsage::IsDigitalSignatureSet Method

Verifies if the digital signature flag is set.

C++

```
mxt_result IsDigitalSignatureSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the digital signature flag is set.

2.12.1.12.3.6 - CKeyUsage::IsEncipherOnlySet Method

Verifies if the encipher only flag is set.

C++

```
mxt_result IsEncipherOnlySet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the encipher only flag is set.

2.12.1.12.3.7 - CKeyUsage::IsKeyAgreementSet Method

Verifies if the key agreement flag is set.

C++

```
mxt_result IsKeyAgreementSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE

- resSI_TRUE

Description

Verifies if the key agreement flag is set.

2.12.1.12.3.8 - CKeyUsage::IsKeyEnciphermentSet Method

Verifies if the key encipherment flag is set.

C++

```
mxt_result IsKeyEnciphermentSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the key encipherment flag is set.

2.12.1.12.3.9 - CKeyUsage::IsNonRepudiationSet Method

Verifies if the non repudiation flag is set.

C++

```
mxt_result IsNonRepudiationSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the non repudiation flag is set.

2.12.1.12.4 - Operators

2.12.1.12.4.1 - CKeyUsage::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CKeyUsage& rKeyUsage) const;
```

Parameters

Parameters	Description
IN const CKeyUsage& rKeyUsage	Reference to the CKeyUsage (See page 718) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.12.4.2 - CKeyUsage::= Operator

Assignment operator.

C++

```
CKeyUsage& operator =( IN const CKeyUsage& rKeyUsage);
```

Parameters

Parameters	Description
IN const CKeyUsage& rKeyUsage	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.12.4.3 - CKeyUsage::== Operator

Comparison operator.

C++

```
bool operator ==( IN const CKeyUsage& rKeyUsage) const;
```

Parameters

Parameters	Description
IN const CKeyUsage& rKeyUsage	Reference to the CKeyUsage (see page 718) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.13 - CNetscapeCertificateType Class

Class to handle the Netscape certificates extensions certificates.

Class Hierarchy**C++**

```
class CNetScapeCertificateType : public CCertificateExtension;
```

Description

CNetScapeCertificateType is the class to handle the Netscape certificates extensions.

Location

Pki/CNetScapeCertificateType.h

See Also

CCertificateExtension (see page 699)

Constructors

Constructor	Description
~CNetScapeCertificateType (see page 724)	Constructor.

CCertificateExtension Class

CCertificateExtension Class	Description
~CCertificateExtension (see page 700)	Constructor.

Legend

Destructors

Destructor	Description
~CNetScapeCertificateType (See page 725)	Destructor.

CCertificateExtension Class

CCertificateExtension Class	Description
~CCertificateExtension (See page 700)	Destructor.

Legend

◆	Method
▼	virtual

Operators

Operator	Description
!= (See page 727)	Different than operator.
= (See page 727)	Assignment operator.
== (See page 728)	Comparison operator.

CCertificateExtension Class

CCertificateExtension Class	Description
!= (See page 703)	Different than operator.
= (See page 703)	Assignment operator.
== (See page 704)	Comparison operator.

Legend

◆	Method
---	--------

Methods

Method	Description
IsClientAuthenticationSet (See page 725)	Verifies if the client authentication flag is set.
IsObjectSignedCertificateAuthoritySet (See page 726)	Verifies if the object signed certificate authority flag is set.
IsObjectSignedSet (See page 726)	Verifies if the object signed flag is set.
IsServerAuthenticationSet (See page 726)	Verifies if the server authentication flag is set.
IsSmimeCertificateAuthoritySet (See page 726)	Verifies if the SMIME certificate authority flag is set.
IsSmimeSet (See page 727)	Verifies if the SMIME flag is set.
IsSslCertificateAuthoritySet (See page 727)	Verifies if the SSL certificate authority flag is set.

CCertificateExtension Class

CCertificateExtension Class	Description
GetBasicConstraints (See page 700)	Gets the Basic Constraints extension.
GetExtendedKeyUsage (See page 701)	Gets the Extended key usage.
GetIssuerAlternateName (See page 701)	Gets the Issuer Alternate Name extension.
GetKeyUsage (See page 701)	Gets the Key usage extension.
GetNetscapeCertificateType (See page 702)	Gets the Netscape certificate extensions.
GetSubjectAlternateName (See page 702)	Gets the Subject Alternate Name extension.
GetType (See page 702)	Gets the type of extension.
IsCritical (See page 703)	Verifies if an extension is critical.

Legend

◆	Method
---	--------

2.12.1.13.1 - Constructors

2.12.1.13.1.1 - CNetScapeCertificateType

2.12.1.13.1.1.1 - CNetScapeCertificateType::CNetScapeCertificateType Constructor

Constructor.

C++

```
CNetscapeCertificateType();
```

Description

Constructor.

2.12.1.13.1.1.2 - CNetscapeCertificateType::CNetscapeCertificateType Constructor

Copy constructor.

C++

```
CNetscapeCertificateType(IN const CNetscapeCertificateType& rNetscapeCertificateType);
```

Parameters

Parameters	Description
IN const CNetscapeCertificateType& rNetscapeCertificateType	Reference to the object to copy.

Description

Constructor.

2.12.1.13.1.1.3 - CNetscapeCertificateType::CNetscapeCertificateType Constructor

Copy constructor.

C++

```
CNetscapeCertificateType(IN const CNetscapeCertificateType* pNetscapeCertificateType);
```

Parameters

Parameters	Description
IN const CNetscapeCertificateType* pNetscapeCertificateType	Pointer to the object to copy.

Description

Constructor.

2.12.1.13.2 - Destructors**2.12.1.13.2.1 - CNetscapeCertificateType::~CNetscapeCertificateType Destructor**

Destructor.

C++

```
virtual ~CNetscapeCertificateType();
```

Description

Destructor.

2.12.1.13.3 - Methods**2.12.1.13.3.1 - CNetscapeCertificateType::IsClientAuthenticationSet Method**

Verifies if the client authentication flag is set.

C++

```
mxt_result IsClientAuthenticationSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the client authentication flag is set.

2.12.1.13.3.2 - CNetscapeCertificateType::IsObjectSignedCertificateAuthoritySet Method

Verifies if the object signed certificate authority flag is set.

C++

```
mxt_result IsObjectSignedCertificateAuthoritySet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the object signed certificate authority flag is set.

2.12.1.13.3.3 - CNetscapeCertificateType::IsObjectSignedSet Method

Verifies if the object signed flag is set.

C++

```
mxt_result IsObjectSignedSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the object signed flag is set.

2.12.1.13.3.4 - CNetscapeCertificateType::IsServerAuthenticationSet Method

Verifies if the server authentication flag is set.

C++

```
mxt_result IsServerAuthenticationSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the server authentication flag is set.

2.12.1.13.3.5 - CNetscapeCertificateType::IsSmimeCertificateAuthoritySet Method

Verifies if the SMIME certificate authority flag is set.

C++

```
mxt_result IsSmimeCertificateAuthoritySet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE

- resSI_TRUE

Description

Verifies if the SMIME certificate authority flag is set.

2.12.1.13.3.6 - CNetscapeCertificateType::IsSmimeSet Method

Verifies if the SMIME flag is set.

C++

```
mxt_result IsSmimeSet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the SMIME flag is set.

2.12.1.13.3.7 - CNetscapeCertificateType::IsSslCertificateAuthoritySet Method

Verifies if the SSL certificate authority flag is set.

C++

```
mxt_result IsSslCertificateAuthoritySet() const;
```

Returns

- resFE_FAIL
- resSI_FALSE
- resSI_TRUE

Description

Verifies if the SSL certificate authority flag is set.

2.12.1.13.4 - Operators

2.12.1.13.4.1 - CNetscapeCertificateType::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CNetscapeCertificateType& rNetscapeCertificateType) const;
```

Parameters

Parameters	Description
IN const CNetscapeCertificateType& rNetscapeCertificateType	Reference to the CNetscapeCertificateType (See page 723) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.13.4.2 - CNetscapeCertificateType::= Operator

Assignment operator.

C++

```
CNetscapeCertificateType& operator =( IN const CNetscapeCertificateType& rNetscapeCertificateType);
```

Parameters

Parameters	Description
IN const CNetscapeCertificateType& rNetscapeCertificateType	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.13.4.3 - CNetscapeCertificateType::== Operator

Comparison operator.

C++

```
bool operator ==( IN const CNetscapeCertificateType& rNetscapeCertificateType) const;
```

Parameters

Parameters	Description
IN const CNetscapeCertificateType& rNetscapeCertificateType	Reference to the CNetscapeCertificateType (see page 723) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.14 - CPkcs12 Class

Class that manages the creation and parsing of PKCS#12 v1.0 formatted buffers.

Class Hierarchy

```
CPkcs12
```

C++

```
class CPkcs12;
```

Description

CPkcs12 is the class that manages the creation and parsing of PKCS#12 v1.0 formatted buffers. PKCS#12 is a file format used to manage personal information. In other words, a personal certificate, a personal private key, and possibly one or more intermediate certificates.

Notes

For this class to work correctly under OpenSSL, RC2 and 3DES must have been included within the library. They should be enabled if no-rc2 and no-des are absent from the command line arguments passed to Configure.

Location

Pki/CPkcs12.h

See Also

CCertificateChain

Constructors

Constructor	Description
CPkcs12 (see page 729)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CPkcs12 (see page 730)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 732)	Assignment operator.

Legend

	Method
---	--------

Methods

Method	Description
 GetCertificateChain (see page 730)	Retrieves the certificate chain information associated with this CPkcs12 instance.
 Restore (see page 730)	Restores a PKCS#12 structure from its serialized state.

Legend

	Method
---	--------

2.12.1.14.1 - Constructors

2.12.1.14.1.1 - CPkcs12

2.12.1.14.1.1.1 - CPkcs12::CPkcs12 Constructor

Constructor.

C++

```
CPkcs12();
```

Description

Constructor.

2.12.1.14.1.1.2 - CPkcs12::CPkcs12 Constructor

Copy constructor.

C++

```
CPkcs12(IN const CPkcs12& rPkcs12);
```

Parameters

Parameters	Description
IN const CPkcs12& rPkcs12	Reference to the object to copy.

Description

Copy constructor.

2.12.1.14.1.1.3 - CPkcs12::CPkcs12 Constructor

Copy constructor.

C++

```
CPkcs12(IN const CPkcs12* pPkcs12);
```

Description

Copy constructor.

2.12.1.14.2 - Destructors

2.12.1.14.2.1 - CPkcs12::~CPkcs12 Destructor

Destructor.

C++

```
virtual ~CPkcs12();
```

Description

Destructor.

2.12.1.14.3 - Methods

2.12.1.14.3.1 - CPkcs12::GetCertificateChain Method

Retrieves the certificate chain information associated with this CPkcs12 (see page 728) instance.

C++

```
mxt_result GetCertificateChain(OUT CCertificateChain* pCertificateChain) const;
```

Parameters

Parameters	Description
OUT CCertificateChain* pCertificateChain	Pointer to contain the certificate chain.

Returns

- resFE_INVALID_ARGUMENT
- resFE_INVALID_STATE
- resS_OK

Description

Gets the certificate chain contained inside this PKCS12 object.

2.12.1.14.3.2 - Restore

2.12.1.14.3.2.1 - CPkcs12::Restore Method

Restores a PKCS#12 structure from its serialized state.

C++

```
mxt_result Restore(IN const CBlob* pblobPkcs12, IN IPassPhrase* pPassPhrase, IN mxt_opaque opqPassPhraseParameter);
```

Parameters

Parameters	Description
IN const CBlob* pblobPkcs12	Pointer to the blob containing the serialized PKCS12 object.
IN IPassPhrase* pPassPhrase	Passphrase interface used to parse the object.
IN mxt_opaque opqPassPhraseParameter	Parameter used to get the passphrase from the passphrase object.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a PKCS12 object from its serialized state.

2.12.1.14.3.2.2 - CPkcs12::Restore Method

Restores a PKCS#12 structure from its serialized state.

C++

```
mxt_result Restore(IN const CBlob* pblobPkcs12, IN const char* pszPassPhrase);
```

Parameters

Parameters	Description
IN const CBlob* pblobPkcs12	Pointer to the blob containing the serialized PKCS12 object.
IN const char* pszPassPhrase	Passphrase used to parse the object.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a PKCS12 object from its serialized state.

2.12.1.14.3.2.3 - CPkcs12::Restore Method

Restores a PKCS#12 structure from its serialized state.

C++

```
mxt_result Restore(IN const uint8_t* puPkcs12, IN unsigned int uPkcs12Size, IN IPassPhrase* pPassPhrase, IN
mxt_opaque opqPassPhraseParameter);
```

Parameters

Parameters	Description
IN const uint8_t* puPkcs12	Pointer to the buffer containing the serialized PKCS12 object.
IN unsigned int uPkcs12Size	Size of the input buffer.
IN IPassPhrase* pPassPhrase	Passphrase interface used to parse the object.
IN mxt_opaque opqPassPhraseParameter	Parameter used to get the passphrase from the passphrase object.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL
- resS_OK

Description

Restores a PKCS12 object from its serialized state.

2.12.1.14.3.2.4 - CPkcs12::Restore Method

Restores a PKCS#12 structure from its serialized state.

C++

```
mxt_result Restore(IN const uint8_t* puPkcs12, IN unsigned int uPkcs12Size, IN const char* pszPassPhrase);
```

Parameters

Parameters	Description
IN const uint8_t* puPkcs12	Pointer to the buffer containing the serialized PKCS12 object.
IN unsigned int uPkcs12Size	Size of the input buffer.
IN const char* pszPassPhrase	Passphrase used to parse the object.

Returns

- resFE_INVALID_ARGUMENT
- resFE_FAIL

- resS_OK

Description

Restores a PKCS12 object from its serialized state.

2.12.1.14.4 - Operators

2.12.1.14.4.1 - CPkcs12::= Operator

Assignment operator.

C++

```
CPkcs12& operator =(IN const CPkcs12& rPkcs12);
```

Parameters

Parameters	Description
IN const CPkcs12& rPkcs12	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.15 - CSubjectAlternateName Class

Class that manages the subjectAlternateName extension.

Class Hierarchy



C++

```
class CSubjectAlternateName : public CALternateName;
```

Description

CSubjectAlternateName is the class that manages the subjectAlternateName extension.

Location

Pki/CSubjectAlternateName.h

Constructors

Constructor	Description
◆ CSubjectAlternateName (See page 733)	Constructor.

CALternateName Class

CALternateName Class	Description
◆ CALternateName (See page 665)	Constructor.

Legend



Destructors

Destructor	Description
◆ ~CSubjectAlternateName (See page 734)	Destructor.

CALternateName Class

CALternateName Class	Description
◆ ~CALternateName (See page 665)	Destructor.

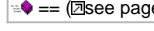
Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 734)	Different than operator.
 = (see page 734)	Assignment operator.
 == (see page 735)	Comparison operator.

CAAlternateName Class

CAAlternateName Class	Description
 != (see page 667)	Different than operator.
 = (see page 668)	Assignment operator.
 == (see page 668)	Comparison operator.

Legend

	Method
--	--------

Methods

CAAlternateName Class

CAAlternateName Class	Description
 GetDnsName (see page 665)	Gets the value of a name of type eNAME_TYPE_DNS_NAME. An error is returned if there is a type mismatch.
 GetIpAddress (see page 666)	Gets the value of a name of type eNAME_TYPE_IP_ADDRESS. An error is returned if there is a type mismatch.
 GetNameCount (see page 666)	Gets the number of name entries.
 GetNameType (see page 666)	Gets the type of a name entry.
 GetRfc822Name (see page 667)	Gets the value of the name of type eNAME_TYPE_RFC_822_NAME (aka email). An error is returned if there is a type mismatch.
 GetUniformResourceIdentifier (see page 667)	Gets the value of a name of type eNAME_TYPE_UNIFORM_RESOURCE_IDENTIFIER. An error is returned if there is a type mismatch.

Legend

	Method
--	--------

2.12.1.15.1 - Constructors

2.12.1.15.1.1 - CSubjectAlternateName

2.12.1.15.1.1.1 - CSubjectAlternateName::CSubjectAlternateName Constructor

Constructor.

C++

```
CSubjectAlternateName( );
```

Description

Constructor.

2.12.1.15.1.1.2 - CSubjectAlternateName::CSubjectAlternateName Constructor

Copy constructor.

C++

```
CSubjectAlternateName( IN const CSubjectAlternateName& rSubjectAlternateName );
```

Parameters

Parameters	Description
IN const CSubjectAlternateName& rSubjectAlternateName	Reference to the CSubjectAlternateName to copy.

Description

Copy constructor.

2.12.1.15.1.1.3 - CSubjectAlternateName::CSubjectAlternateName Constructor

Copy constructor.

C++

```
CSubjectAlternateName( IN const CSubjectAlternateName* pSubjectAlternateName );
```

Parameters

Parameters	Description
IN const CSubjectAlternateName* pSubjectAlternateName	Pointer to the CSubjectAlternateName to copy.

Description

Copy constructor.

2.12.1.15.2 - Destructors

2.12.1.15.2.1 - CSubjectAlternateName::~CSubjectAlternateName Destructor

Destructor.

C++

```
virtual ~CSubjectAlternateName( );
```

Description

Destructor.

2.12.1.15.3 - Operators

2.12.1.15.3.1 - CSubjectAlternateName::!= Operator

Different than operator.

C++

```
bool operator !=( IN const CSubjectAlternateName& rSubjectAlternateName ) const;
```

Parameters

Parameters	Description
IN const CSubjectAlternateName& rSubjectAlternateName	Reference to the CSubjectAlternateName (See page 732) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.15.3.2 - CSubjectAlternateName::== Operator

Assignment operator.

C++

```
CSubjectAlternateName& operator =( IN const CSubjectAlternateName& rSubjectAlternateName );
```

Parameters

Parameters	Description
IN const CSubjectAlternateName& rSubjectAlternateName	Reference to the CSubjectAlternateName (see page 732) to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.15.3.3 - CSubjectAlternateName::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CSubjectAlternateName& rSubjectAlternateName) const;
```

Parameters

Parameters	Description
IN const CSubjectAlternateName& rSubjectAlternateName	Reference to the CSubjectAlternateName (see page 732) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.12.1.16 - CSubjectKeyIdentifier Class

Class used to handle the subject key identifier extensions.

Class Hierarchy



C++

```
class CSubjectKeyIdentifier : public MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME;
```

Description

CSubjectKeyIdentifier is the class to handle the subject key identifier certificates extensions. These extensions contain information to identify a certificate that contains the public key used for this certificate.

Location

Pki/CSubjectKeyIdentifier.h

See Also

CCertificateExtension (see page 699)

Constructors

Constructor	Description
~CSubjectKeyIdentifier (see page 736)	Constructor.

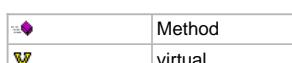
Legend



Destructors

Destructor	Description
~CSubjectKeyIdentifier (see page 737)	Destructor.

Legend



Operators

Operator	Description
<code>!=</code> (see page 737)	Different than operator.
<code>=</code> (see page 737)	Assignment operator.
<code>==</code> (see page 738)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
<code>GetSubjectKeyIdentifier</code> (see page 737)	Gets the subject key identifier.

Legend

	Method
--	--------

2.12.1.16.1 - Constructors

2.12.1.16.1.1 - CSubjectKeyIdentifier

2.12.1.16.1.1.1 - CSubjectKeyIdentifier::CSubjectKeyIdentifier Constructor

Constructor.

C++

```
CSubjectKeyIdentifier();
```

Description

Constructor.

2.12.1.16.1.1.2 - CSubjectKeyIdentifier::CSubjectKeyIdentifier Constructor

Copy constructor.

C++

```
CSubjectKeyIdentifier(IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier);
```

Parameters

Parameters	Description
IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier	Reference to the object to copy.

Description

Constructor.

2.12.1.16.1.1.3 - CSubjectKeyIdentifier::CSubjectKeyIdentifier Constructor

Copy constructor.

C++

```
CSubjectKeyIdentifier(IN const CSubjectKeyIdentifier* pSubjectKeyIdentifier);
```

Parameters

Parameters	Description
IN const CSubjectKeyIdentifier* pSubjectKeyIdentifier	Pointer to the object to copy.

Description

Constructor.

2.12.1.16.2 - Destructors

2.12.1.16.2.1 - CSubjectKeyIdentifier::~CSubjectKeyIdentifier Destructor

Destructor.

C++

```
virtual ~CSubjectKeyIdentifier();
```

Description

Destructor.

2.12.1.16.3 - Methods

2.12.1.16.3.1 - CSubjectKeyIdentifier::GetSubjectKeyIdentifier Method

Gets the subject key identifier.

C++

```
mxt_result GetSubjectKeyIdentifier(OUT CBlob* pblobIdentifier) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobIdentifier	Pointer to the CBlob (see page 95) object to contain the return value.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT

Description

Gets the Subject Key Identifier from the extension.

2.12.1.16.4 - Operators

2.12.1.16.4.1 - CSubjectKeyIdentifier::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier) const;
```

Parameters

Parameters	Description
IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier	Reference to the CSubjectKeyIdentifier (see page 735) to compare.

Returns

True if both objects are different, false otherwise.

Description

Verifies if both objects are different.

2.12.1.16.4.2 - CSubjectKeyIdentifier::= Operator

Assignment operator.

C++

```
CSubjectKeyIdentifier& operator =(IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier);
```

Parameters

Parameters	Description
IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier	Reference to the object to assign.

Returns

A reference to the assigned object.

Description

Assigns the right hand object to the left hand one.

2.12.1.16.4.3 - CSubjectKeyIdentifier::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier) const;
```

Parameters

Parameters	Description
IN const CSubjectKeyIdentifier& rSubjectKeyIdentifier	Reference to the CSubjectKeyIdentifier (see page 735) to compare.

Returns

True if both objects are equal, false otherwise.

Description

Verifies if both objects are equal.

2.13 - RegExp (Regular Expressions)

This section documents the Sources/RegExp folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 738)
- Structures (see page 740)

2.13.1 - Classes

This section documents the classes of the Sources/RegExp folder.

Classes

Class	Description
CRegExp (see page 738)	This class handles POSIX Extended Regular Expressions.

2.13.1.1 - CRegExp Class

This class handles POSIX Extended Regular Expressions.

Class Hierarchy

```
CRegExp
```

C++

```
class CRegExp;
```

Description

This class abstracts calls to a regular expression library that supports POSIX Extended Regular Expressions.

Location

RegExp/CRegExp.h

Methods

Method	Description
• Find (see page 739)	Performs a simple validation that a string matches a regular expression.
• FindAndReplace (see page 739)	Finds regular expression patterns and formats them following a specified format.
• FindSubExpressions (see page 740)	Finds regular expression patterns and returns sub-expression matches.

Legend**2.13.1.1.1 - Methods****2.13.1.1.1.1 - CRegExp::Find Method**

Performs a simple validation that a string matches a regular expression.

C++

```
static mxt_result Find(IN const char* pszRegExp, IN bool bCaseSensitivity, IN const char* pszString);
```

Parameters

Parameters	Description
IN const char* pszRegExp	C-style string representing the regular expression to use.
IN bool bCaseSensitivity	<ul style="list-style-type: none"> • true: the regular expression is case-sensitive. • false: the regular expression is case-insensitive.
IN const char* pszString	C-style string representing the string on which to apply the regular expression.

Returns

- resS_OK: pszString matches the regular expression.
- resFE_FAIL: pszString does not match the regular expression.
- resFE_INVALID_ARGUMENT: pszRegExp and/or pszString are NULL.

Description

This method validates that a string supplied in pszString matches a regular expression specified in pszRegExp. If the string matches the regular expression, resS_OK is returned. Otherwise, resFE_FAIL is returned.

See Also

[FindSubExpressions](#) (see page 740), [FindAndReplace](#) (see page 739)

2.13.1.1.1.2 - CRegExp::FindAndReplace Method

Finds regular expression patterns and formats them following a specified format.

C++

```
static mxt_result FindAndReplace(IN const char* pszRegExp, IN bool bCaseSensitivity, IN const char* pszString,
IN const char* pszFormat, OUT CString* pstrResult);
```

Parameters

Parameters	Description
IN const char* pszRegExp	C-style string representing the regular expression to use.
IN bool bCaseSensitivity	<ul style="list-style-type: none"> • true: the regular expression is case-sensitive. • false: the regular expression is case-insensitive.
IN const char* pszString	C-style string representing the string on which to apply the regular expression.
IN const char* pszFormat	C-style string representing the format in which to put the sub-expressions found.
OUT CString* pstrResult	CString (see page 126) to hold the formatted result.

Returns

- resS_OK: pszString matches the regular expression.
- resFE_FAIL: pszString does not match the regular expression.
- resFE_INVALID_ARGUMENT: pszRegExp, pszString, pszFormat, and/or pstrResult are NULL.

Description

This method validates that a string supplied in pszString matches a regular expression specified in pszRegExp. If the string matches the regular expression, resS_OK is returned, and the sub-expressions found are formatted according to pszFormat. Otherwise, resFE_FAIL is returned. It is possible to back-reference matched sub-expressions in pszFormat. To do so, the " character, followed by a number

from 1 through 9, must appear in pszFormat. For example, the regular expression (A(B(C)DE)(F)G) has the following sub-expressions that can be used via back-references:

```
1 = ABCDEFG
2 = BCDE
3 = C
4 = F
5 to 9 = error, no matching sub-expression.
```

See Also

[Find](#) (see page 739), [FindSubExpressions](#) (see page 740)

2.13.1.1.1.3 - CRegExp::FindSubExpressions Method

Finds regular expression patterns and returns sub-expression matches.

C++

```
static mxt_result FindSubExpressions(IN const char* pszRegExp, IN bool bCaseSensitivity, IN const char* pszString, IN unsigned int uSubExpCapacity, OUT unsigned int* puSubExpSize, OUT SSubExpression* pastSubExpressions);
```

Parameters

Parameters	Description
IN const char* pszRegExp	C-style string representing the regular expression to use.
IN bool bCaseSensitivity	<ul style="list-style-type: none"> • true: the regular expression is case-sensitive. • false: the regular expression is case-insensitive.
IN const char* pszString	C-style string representing the string on which to apply the regular expression.
IN unsigned int uSubExpCapacity	The capacity of the pastSubExpressions array.
OUT unsigned int* puSubExpSize	The number of SSubExpression (see page 740) structures put in the pastSubExpressions array.
OUT SSubExpression* pastSubExpressions	Pointer to the first element in an array of SSubExpression (see page 740) structures that contain information on sub-expressions on return.

Returns

- resS_OK: pszString matches the regular expression.
- resFE_FAIL: pszString does not match the regular expression.
- resFE_INVALID_ARGUMENT: pszRegExp, pszString, and/or puSubExpSize are NULL, or pastSubExpressions is NULL while uSubExpCapacity is not 0.

Description

This method validates that a string supplied in pszString matches a regular expression specified in pszRegExp. If the string matches the regular expression, resS_OK is returned and information on sub-expressions is put in the array specified by pastSubExpressions. Otherwise, resFE_FAIL is returned. Note that the first sub-expression is always a match on the whole regular expression. Moreover, there is a maximum of 9 sub-expressions per regular expression, not counting the match on the whole regular expression.

See Also

[Find](#) (see page 739), [FindAndReplace](#) (see page 739)

2.13.2 - Structures

This section documents the structures of the Sources/RegExp folder.

Structs

Struct	Description
SSubExpression (see page 740)	Structure representing a sub-expression of a regular expression.

2.13.2.1 - CRegExp::SSubExpression Struct

Structure representing a sub-expression of a regular expression.

C++

```
struct SSubExpression {
    char* pcExpression;
    unsigned int uExpressionSize;
```

};

Members

Members	Description
char* pcExpression;	The first character in a sub-expression.
unsigned int uExpressionSize;	The sub-expression's size.

2.14 - Resolver

This section documents the Sources/Resolver folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 741)
- Structures (see page 762)

2.14.1 - Classes

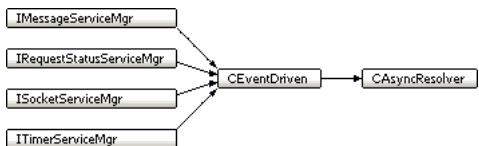
This section documents the classes of the Sources/Resolver folder.

Classes

Class	Description
CAsyncResolver (see page 741)	Singleton asynchronous interface for DNS queries.
CDnsPacket (see page 747)	This class abstracts a DNS packet.
CHostFile (see page 749)	This class servers as a local host file.
CPortableResolver (see page 753)	This is the portable implementation of the resolver core.
CResolver (see page 756)	Synchronisation wrapper for DNS requests.
CResolverCache (see page 759)	This class provides an abstraction to a DNS resolver cache.
IAsyncResolverUser (see page 760)	Interface of the asynchronous resolver user.

2.14.1.1 - CAsyncResolver Class

Singleton asynchronous interface for DNS queries.

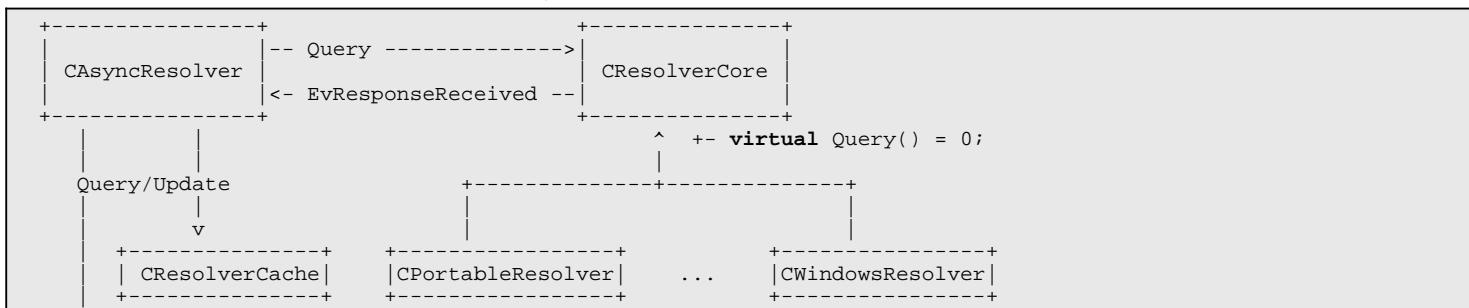
Class Hierarchy**C++**

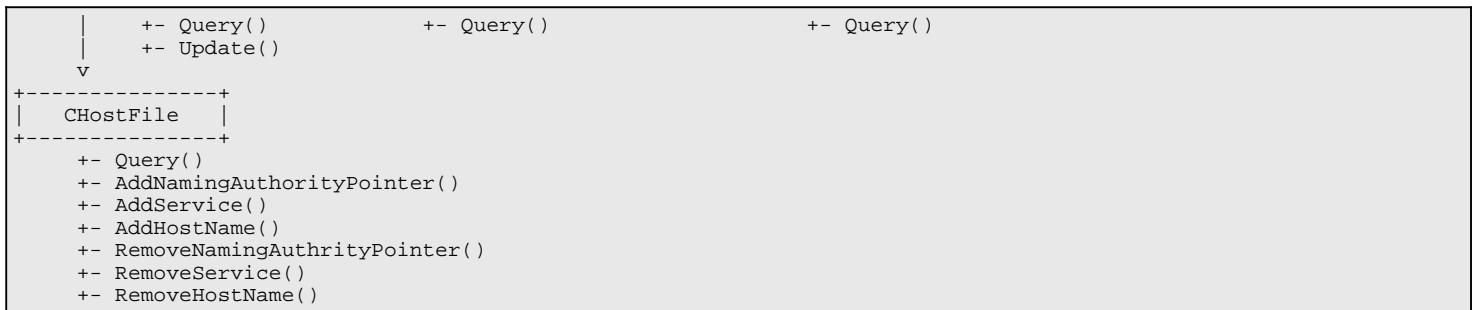
```
class CAsyncResolver : protected CEventDriven;
```

Description

This class provides a singleton asynchronous interface for DNS queries. It also provides query recursion. Notifications are sent on the IAsyncResolverUser (see page 760) interface.

This class performs the generic operations of a DNS resolver and interacts with implementations specific to either the operating system's DNS API or the portable resolver, which uses the Network socket API to perform queries. It uses its own thread, so requests do not block the user's thread. It also launches the core resolver in its own separated thread when it uses a synchronous API (e.g. Operating system's API). This way, queries can be queued while waiting for an answer from the nameserver. Another way to improve the resolver's efficiency is to piggyback queries to others which have the same question. For example, if two equal queries are performed for the same domain name, the first query is sent on the network and the second is only queued waiting for the answer to the first one. Once the answer is received for the query, both queries are resolved at the same time.





Note on support for recursion: The recursion is handled by making successive queries to the DNS and merging the records back in the original response. The latter response is searched every time until no recursions are necessary and therefore all the data necessary to answer the DNS question have been queried. For example:

- let's say an A query returns a single CNAME record. The response is searched for the A record associated with the query name and only a CNAME will be found. The response is saved as the original response and a recursive A query is made to resolve the alias.
- now let's say the A query for the alias has returned an A record (obviously containing an IPv4 address).
- the A record for the alias is merged into the original response (so far only containing a CNAME record).
- the response is then searched for the A record of the original query name and the answer will be found.

Location

`Resolver/CAsyncResolver.h`

See Also

`IAsyncResolverUser` (see page 760), `CResolverCore`, `CResolverCache` (see page 759)

Constructors

CEventDriven Class

CEventDriven Class	Description
◆ CEventDriven (see page 764)	Constructor.

Legend

◆	Method
---	--------

Destructors

CEventDriven Class

CEventDriven Class	Description
◆ ~CEventDriven (see page 764)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
◆ Cancel (see page 743)	Synchronously cancels a query.
◆ DisableCacheNonAuthoritativeResponse (see page 744)	Disallow to store non-authoritative responses in cache.
◆ EnableCacheNonAuthoritativeResponse (see page 744)	Allows to store non-authoritative responses in cache.
◆ EvQueryCanceledA (see page 744)	Notifies that the query has been canceled.
◆ EvResponseReceivedA (see page 744)	Called by the resolver implementation when a query is answered.
◆ GetEnumUrisA (see page 745)	Generates an ENUM query for the specified services and E.164 number.
◆ GetHostByAddressA (see page 745)	Gets the hosts name using its address.
◆ GetHostByNameA (see page 745)	Gets the hosts address using its name.
◆ GetHostFile (see page 746)	Gets the host file so it can be configured.

• GetInstance (see page 746)	Returns the unique instance of this class.
• GetNameServers (see page 746)	&>
• GetNamingAuthorityPointersA (see page 746)	Queries the domain for a list of naming authority pointers.
• GetServicesA (see page 747)	Queries hosts providing the desired service.
• SetNameServers (see page 747)	&>

CEventDriven Class

CEventDriven Class	Description
• Activate (see page 764)	Associates a Servicing Thread with this Event Driven.
• DisableCompletionDetection (see page 765)	Disables the detection of request completion.
• DisableEventsDetection (see page 765)	Disables the detection of events.
• EnableCompletionDetection (see page 766)	Enables the detection of request completion.
• EnableEventsDetection (see page 766)	Enables the detection of events.
• FinalizeAndReleaseA (see page 766)	Finalizes and releases an Event Driven.
• GetIEComUnknown (see page 766)	Returns a pointer to the Servicing Thread. AddRef is already called.
• GetServicingThread (see page 767)	Returns a pointer to the Servicing Thread. AddRef is already called.
• IsCurrentExecutionContext (see page 767)	Returns whether or not the code is executing within the current execution context.
• PostMessage (see page 767)	Pushes a new message into the message queue.
• RegisterRequestStatus (see page 768)	Registers a request status.
• RegisterSocket (see page 768)	Registers a socket.
• Release (see page 768)	Releases an Event Driven.
• StartTimer (see page 769)	Starts a new linear timer.
• StopAllTimers (see page 770)	Stops all timers owned by a manager.
• StopTimer (see page 770)	Stops a timer owned by a manager.
• UnregisterRequestStatus (see page 770)	Unregisters a request status.
• UnregisterSocket (see page 771)	Unregisters a socket.

ITimerServiceMgr Class

ITimerServiceMgr Class	Description
• A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

ISocketServiceMgr Class

ISocketServiceMgr Class	Description
• A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

IRequestStatusServiceMgr Class

IRequestStatusServiceMgr Class	Description
• A EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
A	abstract

2.14.1.1.1 - Methods

2.14.1.1.1.1 - CAyncResolver::Cancel Method

Synchronously cancels a query.

C++

```
mxt_result Cancel(IN CDnsPacket::SQuestion& rstQuestion, IN IAsyncResolverUser* pUser, IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN CDnsPacket::SQuestion& rstQuestion	The question related to the query to cancel.
IN IAsyncResolverUser* pUser	The IAsyncResolverUser (see page 760) interface that queried the resolver.

Returns

- resS_OK: the query has been canceled.
- resFE_INVALID_ARGUMENT: pUser is NULL.

Description

Attempts to synchronously cancel a query before it is sent to the nameserver. If it is active (i.e. Waiting for an answer), the notification is not made.

2.14.1.1.1.2 - CAyncResolver::DisableCacheNonAuthoritativeResponse Method

Disallow to store non-authoritative responses in cache.

C++

```
void DisableCacheNonAuthoritativeResponse();
```

Description

Disallow to store non-authoritative responses in cache. By default, non-authoritative responses are cached. If MXD_RESOLVER_CACHE_ENABLE_SUPPORT (see page 304) is not defined, calling this method has no effect.

2.14.1.1.1.3 - CAyncResolver::EnableCacheNonAuthoritativeResponse Method

Allows to store non-authoritative responses in cache.

C++

```
void EnableCacheNonAuthoritativeResponse();
```

Description

Allows to store non-authoritative responses in cache. By default, non-authoritative responses are cached. If MXD_RESOLVER_CACHE_ENABLE_SUPPORT (see page 304) is not defined, calling this method has no effect.

2.14.1.1.1.4 - CAyncResolver::EvQueryCanceledA Method

Notifies that the query has been canceled.

C++

```
void EvQueryCanceledA(IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	The opaque parameter passed to the resolver core when queried. This opaque value contains a pointer to an internal structure in order to properly identify a query and its state.

Description

Updates the cache when enabled and gets the IAsyncResolverUser (see page 760) notified.

2.14.1.1.1.5 - CAyncResolver::EvResponseReceivedA Method

Called by the resolver implementation when a query is answered.

C++

```
void EvResponseReceivedA(IN CDnsPacket& rResponse, IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN CDnsPacket& rResponse	The response.

IN mxt_opaque opq	The opaque parameter passed to the resolver core when queried. This opaque value contains a pointer to an internal structure in order to properly identify a query and its state.
-------------------	---

Description

Updates the cache when enabled and gets the IAsyncResolverUser (see page 760) notified.

2.14.1.1.1.6 - CAsyncResolver::GetEnumUrisA Method

Generates an ENUM query for the specified services and E.164 number.

C++

```
mxt_result GetEnumUrisA(IN const CString& rstrNumber, IN const CList<CDnsPacket::SEnumServiceData>& rlstSupportedServices, IN CString& rstrZone, IN IAsyncResolverUser* pUser, IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN const CString& rstrNumber	The E.164 number to query.
IN const CList<CDnsPacket::SEnumServiceData>& rlstSupportedServices	The list of supported services. Only URIs for the supported services are returned.
IN CString& rstrZone	The zone to query. If empty, it defaults to e164.arpa.
IN IAsyncResolverUser* pUser	The user to notify upon query completion.
IN mxt_opaque opq	The opaque parameter to return with the response.

Returns

- resS_OK: the query has been posted.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Converts rstrNumber into an Application Unique String (AUS) and an ENUM FQDN and recursively queries for URIs associated with the latter on the given zone as per RFC 3761.

2.14.1.1.1.7 - CAsyncResolver::GetHostByAddressA Method

Gets the hosts name using its address.

C++

```
mxt_result GetHostByAddressA(IN const CSocketAddr& rAddress, IN IAsyncResolverUser* pUser, IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddress	The address to query.
IN IAsyncResolverUser* pUser	The user to notify upon query completion.
IN mxt_opaque opq	The opaque parameter to return with the response.

Returns

- resS_OK: the query has been posted.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries for the names associated with rAddress, as per RFC 1035.

2.14.1.1.1.8 - CAsyncResolver::GetHostByNameA Method

Gets the hosts address using its name.

C++

```
mxt_result GetHostByNameA(IN const CString& rstrName, IN IAsyncResolverUser* pUser, IN mxt_opaque opq, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrName	The name to query.
IN IAsyncResolverUser* pUser	The user to notify upon query completion.
IN mxt_opaque opq	The opaque parameter to return with the response.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the addresses to return.

Returns

- resS_OK: the query has been posted.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Recursively queries for the addresses associated with rstrName, as per RFC 1035. Note that only addresses of the given family will be returned.

2.14.1.1.9 - CAsyncResolver::GetHostFile Method

Gets the host file so it can be configured.

C++

```
CHostFile* GetHostFile();
```

Returns

A pointer to the CHostFile (see page 749) of the resolver.

Description

Gets the CHostFile (see page 749) of the resolver.

2.14.1.1.10 - CAsyncResolver::GetInstance Method

Returns the unique instance of this class.

C++

```
static CAsyncResolver* GetInstance();
```

Returns

The unique instance of this class.

Description

Returns the unique instance of this class.

2.14.1.1.11 - CAsyncResolver::GetNameServers Method

&>

C++

```
void GetNameServers(OUT CList<CSocketAddr>& rlstServers);
```

2.14.1.1.12 - CAsyncResolver::GetNamingAuthorityPointersA Method

Queries the domain for a list of naming authority pointers.

C++

```
mxt_result GetNamingAuthorityPointersA(IN const CString& rstrDomain, IN IAsyncResolverUser* pUser, IN mxt_opaque opq, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrDomain	The domain to query.
IN IAsyncResolverUser* pUser	The user to notify upon query completion.
IN mxt_opaque opq	The opaque parameter to return with the response.

IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the addresses to return.
---	--

Returns

- resS_OK: the query has been posted.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries rstrDomain for naming authority pointers, as per RFCs 3401, 3402 and 3403. For compatibility reasons with SIP (RFC 3261), the 's' flag specified in the obsoleted RFC 2915 is supported. This specification stated that the 's' flag meant that the replacement field was a service (an SRV record). SRV records are recursively retrieved.

2.14.1.1.13 - CAsyncResolver::GetServicesA Method

Queries hosts providing the desired service.

C++

```
mxt_result GetServicesA(IN const CString& rstrService, IN IAsyncResolverUser* pUser, IN mxt_opaque opq, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrService	The service to query in the form ...domain.
IN IAsyncResolverUser* pUser	The user to notify upon query completion.
IN mxt_opaque opq	The opaque parameter to return with the response.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the addresses to return.

Returns

- resS_OK: the query has been posted.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Recursively queries for the service associated with rstrService, as per RFC 2782. Note that IPv6 addresses for hosts may be returned, according to the DNS database configuration.

Note that there is no validation on the format of rstrService.

2.14.1.1.14 - CAsyncResolver::SetNameServers Method

&>

C++

```
mxt_result SetNameServers(IN CList<CSocketAddr>& rlstServers);
```

2.14.1.2 - CDnsPacket Class

This class abstracts a DNS packet.

Class Hierarchy

```
CDnsPacket
```

C++

```
class CDnsPacket;
```

Description

This class abstracts a DNS packet.

Location

Resolver/CDnsPacket.h

Constructors

Constructor	Description
CDnsPacket (see page 748)	Default Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CDnsPacket (see page 748)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 749)	Assignment Operator.

Legend

	Method
--	--------

Methods

Method	Description
 ReleaseExpiredRecords (see page 749)	This is ReleaseExpiredRecords, a member of the class CDnsPacket.
 ReleaseRecord (see page 749)	This is ReleaseRecord, a member of the class CDnsPacket.
 UpdateRecords (see page 749)	Copies the records from an existing packet.

Legend

	Method
--	--------

2.14.1.2.1 - Constructors**2.14.1.2.1.1 - CDnsPacket****2.14.1.2.1.1.1 - CDnsPacket::CDnsPacket Constructor**

Default Constructor.

C++

```
CDnsPacket();
```

Description

Default constructor.

2.14.1.2.1.1.2 - CDnsPacket::CDnsPacket Constructor

Copy Constructor.

C++

```
CDnsPacket(IN const CDnsPacket& rSrc);
```

2.14.1.2.2 - Destructors**2.14.1.2.2.1 - CDnsPacket::~CDnsPacket Destructor**

Destructor.

C++

```
virtual ~CDnsPacket();
```

Description

Destructor.

2.14.1.2.3 - Methods

2.14.1.2.3.1 - CDnsPacket::ReleaseExpiredRecords Method

This is ReleaseExpiredRecords, a member of the class CDnsPacket.

C++

```
void ReleaseExpiredRecords();
```

2.14.1.2.3.2 - CDnsPacket::ReleaseRecord Method

This is ReleaseRecord, a member of the class CDnsPacket.

C++

```
static void ReleaseRecord(INOUT SRecord* pstRecord);
```

2.14.1.2.3.3 - CDnsPacket::UpdateRecords Method

Copies the records from an existing packet.

C++

```
void UpdateRecords(IN const CDnsPacket& rSrc);
```

Parameters

Parameters	Description
IN const CDnsPacket& rSrc	The source of the records.

Description

Copies the records from the source and recomputes the TTL if applicable.

2.14.1.2.4 - Operators

2.14.1.2.4.1 - CDnsPacket::= Operator

Assignment Operator.

C++

```
CDnsPacket& operator =(IN const CDnsPacket& rSrc);
```

2.14.1.2.5 - Friends

2.14.1.2.5.1 - friend class CHostFile Friend

This is friend class CHostFile.

C++

```
friend class CHostFile;
```

2.14.1.2.5.2 - friend class CResolverCache Friend

```
friend class CResolverCache;
```

2.14.1.3 - CHostFile Class

This class servers as a local host file.

Class Hierarchy

```
[CHostFile]
```

C++

```
class CHostFile;
```

Description

Implements a local host file. This host file is used to configure DNS records locally when it is impossible to do so on the DNS server.

This is different from the resolver cache as this is static. Once an item is added / removed it is done so for the lifetime of the resolver. The TTL value of all host file records is hardcoded to 0xFFFFFFFF.

The host file record has precedence over the DNS cache and the resolver. All types of queries can be configured into the host file. Queries are configured step by step.

Location

Resolver/CHostFile.h

See Also

CAsyncResolver (see page 741)

Example

To configure a NAPTR record containing 2 SRV records with each a single address:

```

CAsyncResolver* pResolver = CAsyncResolver::GetInstance();
CHostFile* pHostFile = pResolver->GetHostFile();

MX_TFW_ASSERT(pHostFile != NULL);

// Add NAPTR records.
CString strNaptrQuestion = "www.private.mediacorp.com";
CVector<CDnsPacket::SNaptrRecordData> vecstNaptrRecord;
CDnsPacket::SNaptrRecordData stNaptrRecordDataTcp;
stNaptrRecordDataTcp.m_uOrder = 1;
stNaptrRecordDataTcp.m_uPreference = 1;
stNaptrRecordDataTcp.m_strServices = "SIP+D2T";
stNaptrRecordDataTcp.m_strFlags = "S";
stNaptrRecordDataTcp.m_strReplacement = "_sip._tcp.private.mediacorp.com";
vecstNaptrRecord.Append(stNaptrRecordDataTcp);

// Build 2nd NAPTR record.
CDnsPacket::SNaptrRecordData stNaptrRecordDataUdp;
stNaptrRecordDataUdp.m_uOrder = 2;
stNaptrRecordDataUdp.m_uPreference = 1;
stNaptrRecordDataUdp.m_strServices = SIP+D2U;
stNaptrRecordDataUdp.m_strFlags = "S";
stNaptrRecordDataUdp.m_strReplacement = "_sip._udp.private.mediacorp.com";
vecstNaptrRecord.Append(stNaptrRecordDataUdp);

mxt_result res = pHostFile->AddNamingAuthorityPointer(strNaptrQuestion,
                                                       vecstNaptrRecord);

CDnsPacket::SSrvRecordData stTcpRecordData;
stTcpRecordData.m_uPort = 5060;
stTcpRecordData.m_uPriority = 1;
stTcpRecordData.m_uWeight = 10;
stTcpRecordData.m_strTarget = "name.tcp.private.mediacorp.com";

res = pHostFile->AddService("_sip._tcp.private.mediacorp.com",
                           stTcpRecordData);

CDnsPacket::SSrvRecordData stUdpRecordData;
stUdpRecordData.m_uPort = 5060;
stUdpRecordData.m_uPriority = 2;
stUdpRecordData.m_uWeight = 10;
stUdpRecordData.m_strTarget = "name.udp.private.mediacorp.com";

res = pHostFile->AddService("_sip._udp.private.mediacorp.com",
                           stUdpRecordData);

CVector<CString> vecstrAddresses;

vecstrAddresses.Append("127.0.0.1");
res = pHostFile->AddHostName("name.udp.private.mediacorp.com",
                           vecstrAddresses,
                           CSocketAddr::eINET);
vecstrAddresses.EraseAll();

vecstrAddresses.Append("127.0.0.2");
vecstrAddresses.Append("127.0.0.22");
res = pHostFile->AddHostName("_sip._tcp.private.mediacorp.com",
                           vecstrAddresses,
                           CSocketAddr::eINET);
vecstrAddresses.EraseAll();

```

Methods

Method	Description
• AddHostName (see page 751)	Adds a A / AAAA record to the host file.
• AddNamingAuthorityPointer (see page 751)	Adds a NAPTR record to the host file.
• AddService (see page 752)	Adds a SRV record to the host file.
• RemoveHostName (see page 752)	Removes a A / AAAA record from the host file.
• RemoveNamingAuthorityPointer (see page 752)	Removes a NAPTR record from the host file.
• RemoveService (see page 753)	Removes a SRV record from the host file.

Legend

	Method
---	--------

2.14.1.3.1 - Methods

2.14.1.3.1.1 - CHostFile::AddHostName Method

Adds a A / AAAA record to the host file.

C++

```
mxt_result AddHostName(IN const CString& rstrHost, IN CVector<CString>& rvecstrAnswerAddresses, IN
CSocketAddr::EAddressFamily eFamily);
```

Parameters

Parameters	Description
IN const CString& rstrHost	The host name.
IN CSocketAddr::EAddressFamily eFamily	The family (IPv4 or IPv6).
rvecaddrAnswerAddresses	A vector of addresses.

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Adds a host name record to the host file with the possible addresses.

See Also

[AddNamingAuthorityPointer](#) ([see page 751](#)), [AddService](#) ([see page 752](#)), [RemoveHostName](#) ([see page 752](#))

2.14.1.3.1.2 - CHostFile::AddNamingAuthorityPointer Method

Adds a NAPTR record to the host file.

C++

```
mxt_result AddNamingAuthorityPointer(IN const CString& rstrDomain, IN CVector<CDnsPacket::SNaptrRecordData>&
rvecstAnswerServices);
```

Parameters

Parameters	Description
IN const CString& rstrDomain	The domain name. Example: example.com
IN CVector<CDnsPacket::SNaptrRecordData>& rvecstAnswerServices	A vector that corresponds to the resolved SRV record names. _sip._udp.example.com is an example.

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Adds a NAPTR (naming authority pointer) record to the host file with the corresponding answers.

Each item of the rvecstAnswerServices is filled by the application. The list of SSrvRecordData in the SNaptrRecordData structure MUST

be left empty.

See Also

AddService (see page 752), AddHostName (see page 751), RemoveNamingAuthorityPointer (see page 752)

2.14.1.3.1.3 - CHostFile::AddService Method

Adds a SRV record to the host file.

C++

```
mxt_result AddService(IN const CString& rstrService, IN CDnsPacket::SSrvRecordData& rstSrvAnswer);
```

Parameters

Parameters	Description
IN const CString& rstrService	The service name. Example: _sip._udp.www.example.com
IN CDnsPacket::SSrvRecordData& rstSrvAnswer	The SRV structure to be returned by the DNS query.

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Adds a SRV (service) record to the host file with the host name.

Each item of the rstSrvAnswer is filled by the application. The list of CSocketAddr (see page 545) in the SSrvRecordData structure MUST be left empty.

See Also

AddNamingAuthorityPointer (see page 751), AddHostName (see page 751), RemoveService (see page 753)

2.14.1.3.1.4 - CHostFile::RemoveHostName Method

Removes a A / AAAA record from the host file.

C++

```
mxt_result RemoveHostName(IN const CString& rstrHost, IN CSocketAddr::EAddressFamily eFamily);
```

Parameters

Parameters	Description
IN const CString& rstrHost	The service name. Example: _sip._udp.www.example.com
IN CSocketAddr::EAddressFamily eFamily	The family (IPv4 or IPv6).
rvecaddrAnswerAddresses	A vector of addresses.

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Removes a host name record from the host file.

A call to RemoveHostName is not recursive and will not remove any NAPTR and SRV records related to this.

See Also

AddHostName (see page 751), RemoveNamingAuthorityPointer (see page 752), RemoveService (see page 753).

2.14.1.3.1.5 - CHostFile::RemoveNamingAuthorityPointer Method

Removes a NAPTR record from the host file.

C++

```
mxt_result RemoveNamingAuthorityPointer(IN const CString& rstrDomain);
```

Parameters

Parameters	Description
IN const CString& rstrDomain	The domain name. Example: www.example.com

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Removes a NAPTR (naming authority pointer) record from the host file.

A call to RemoveNamingAuthorityPointer is not recursive and will not remove any SRV and A/AAAA records related to this.

See Also

AddNamingAuthorityPointer (see page 751), RemoveService (see page 753), RemoveHostName (see page 752)

2.14.1.3.1.6 - CHostFile::RemoveService Method

Removes a SRV record from the host file.

C++

```
mxt_result RemoveService(IN const CString& rstrService);
```

Parameters

Parameters	Description
IN const CString& rstrService	The service name. Example: _sip._udp.www.example.com

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Removes a SRV (service) record from the host file.

A call to RemoveService is not recursive and will not remove any NAPTR and A/AAAA records related to this.

See Also

AddService (see page 752), RemoveNamingAuthorityPointer (see page 752), RemoveHostName (see page 752)

2.14.1.3.2 - Friends

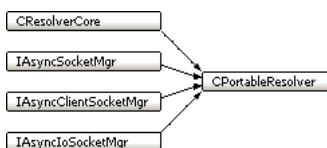
2.14.1.3.2.1 - friend class CAsyncResolver Friend

```
friend class CAsyncResolver;
```

2.14.1.4 - CPortableResolver Class

This is the portable implementation of the resolver core.

Class Hierarchy



C++

```
class CPortableResolver : public CResolverCore, public IAsyncSocketMgr, public IAsyncClientSocketMgr, public IAsyncIoSocketMgr;
```

Description

This class implements a portable resolver, which uses the Framework asynchronous socket API to send and receive DNS requests. It queues queries as they come in and sends them when the socket is ready. If a query comes in and there is an identical one waiting for a response, it just bundles itself with the latter so it saves on sending multiple identical queries on the network. As recommended in RFC 1035, if a query fails (including timing out), it sends it to the next nameserver, prior to retrying on the nameserver that caused the failure to occur.

The portable resolver uses a list of nameservers where to send queries. This list is considered ordered by decreasing order of priority. If a query to a nameserver fails (times out after MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS (see page 303)), it switches to the subsequent nameserver and so on.

If all the nameservers fail, it then wraps around and goes through the list again. This is done MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS (see page 303) times. This is done independently for each query (transaction). This means that every transaction will begin by trying the top-priority nameserver first. This encourages using the primary nameserver.

Finally, if an ICMP error occurs while either sending or receiving on a nameserver for a given transaction, all ongoing transactions that use this nameserver switch.

Location

Resolver/CPortableResolver.h

Methods

Method	Description
EvAsyncClientSocketMgrBound (see page 754)	
EvAsyncClientSocketMgrConnected (see page 754)	Notifies of a successful connection to a remote server.
EvAsyncSocketMgrReadyToRecv (see page 755)	
EvAsyncSocketMgrReadyToSend (see page 755)	Notifies the socket user that the socket is ready to send data on the network.
EvAsyncSocketMgrClosed (see page 755)	
EvAsyncSocketMgrClosedByPeer (see page 755)	Notifies of the closure of the socket by the peer.
EvAsyncSocketMgrErrorDetected (see page 756)	
QueryA (see page 756)	

IAsyncSocketMgr Class

IAsyncSocketMgr Class	Description
EvAsyncSocketMgrReadyToRecv (see page 606)	Notifies the socket user that there is data available to be received on the socket.
EvAsyncSocketMgrReadyToSend (see page 606)	Notifies the socket user that the socket is ready to send data on the network.

IAsyncClientSocketMgr Class

IAsyncClientSocketMgr Class	Description
EvAsyncClientSocketMgrBound (see page 601)	Notifies that the client socket has been bound.
EvAsyncClientSocketMgrConnected (see page 602)	Notifies of a successful connection to a remote server.

IAsyncSocketMgr Class

IAsyncSocketMgr Class	Description
EvAsyncSocketMgrClosed (see page 625)	Notifies of the closure of the socket.
EvAsyncSocketMgrClosedByPeer (see page 625)	Notifies of the closure of the socket by the peer.
EvAsyncSocketMgrErrorDetected (see page 625)	Reports an error detected on the socket.

Legend

Method	
abstract	

2.14.1.4.1 - Methods

2.14.1.4.1.1 - CPortableResolver::EvAsyncClientSocketMgrBound Method

```
void EvAsyncClientSocketMgrBound(IN mxt_opaque opq, IN CSocketAddr* pEffectiveLocalAddress);
```

2.14.1.4.1.2 - CPortableResolver::EvAsyncClientSocketMgrConnected Method

Notifies of a successful connection to a remote server.

C++

```
void EvAsyncClientSocketMgrConnected(IN mxt_opaque opq);
```

Parameters

Parameters	Description
opq	Opaque value associated with the socket that was connecting.

Description

This is the event generated by the asynchronous client socket that is connecting to a remote server. This event is generated upon a successful completion of the connection.

Unsuccessful connection attempts are reported through the EvAsyncSocketMgrErrorDetected event of the IAsyncSocketMgr interface.

See Also

IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected

2.14.1.4.1.3 - CPortableResolver::EvAsyncIoSocketMgrReadyToRecv Method

```
void EvAsyncIoSocketMgrReadyToRecv(IN mxt_opaque opq);
```

2.14.1.4.1.4 - CPortableResolver::EvAsyncIoSocketMgrReadyToSend Method

Notifies the socket user that the socket is ready to send data on the network.

C++

```
void EvAsyncIoSocketMgrReadyToSend(IN mxt_opaque opq);
```

Parameters

Parameters	Description
opq	Opaque parameter associated to the socket.

Description

This event is reported by a socket when it is possible for its user to send data on the network. Thus, after receiving this, the socket user should be able to use the various flavors of IAsyncIoSocket::Send and IAsyncUnconnectedIoSocket::SendTo to send the data on the network.

Note that there might be some instances of sockets (depending on the protocol) that do generate this event even if a call to IAsyncIoSocket::Send and IAsyncUnconnectedIoSocket::SendTo will fail with eFE_SOCK_WOULDBLOCK and zero bytes written. The socket user must be able to handle this case.

See Also

IAsyncIoSocket::Send@CBlob* IAsyncIoSocket::Send@uint8_t*, unsigned int IAsyncIoSocket::SendTo@CBlob*, CSocketAddr*
IAsyncIoSocket::SendTo@uint8_t*, unsigned int, CSocketAddr*

2.14.1.4.1.5 - CPortableResolver::EvAsyncSocketMgrClosed Method

```
void EvAsyncSocketMgrClosed(IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN mxt_opaque opq	The socket index.

Description

Releases the socket's interfaces and if all sockets are released, deletes this.

2.14.1.4.1.6 - CPortableResolver::EvAsyncSocketMgrClosedByPeer Method

Notifies of the closure of the socket by the peer.

C++

```
void EvAsyncSocketMgrClosedByPeer(IN mxt_opaque opq);
```

Parameters

Parameters	Description
opq	Opaque value associated with this socket that has been closed.

Description

This event is reported by the socket after it has been closed by the peer to which it was connected.

Note that this can be reported for sockets of type eSTREAM and eSEQPACKET.

See Also

IAsyncSocket::CloseA

2.14.1.4.1.7 - CPortableResolver::EvAsyncSocketMgrErrorDetected Method

```
void EvAsyncSocketMgrErrorDetected(IN mxt_opaque opq, IN mxt_result res);
```

Parameters

Parameters	Description
IN mxt_opaque opq	The socket index.
IN mxt_result res	The socket error.

Description

Switches nameserver for all transactions that used this socket.

2.14.1.4.1.8 - CPortableResolver::QueryA Method

```
mxt_result QueryA(IN CDnsPacket::SQuestion& rstQuestion, IN mxt_opaque opq);
```

2.14.1.4.2 - Friends

2.14.1.4.2.1 - friend class CAyncResolver Friend

```
friend class CAyncResolver;
```

2.14.1.5 - CResolver Class

Synchronisation wrapper for DNS requests.

Class Hierarchy

```
CResolver
```

C++

```
class CResolver;
```

Description

This class is a wrapper around the asynchronous resolver to block on asynchronous requests. It blocks on a different semaphore for every DNS request. Therefore only the current thread is blocked on its request. Other threads may issue requests and will block on different semaphores.

All methods are static, meaning there is no such thing as a 'synchronous resolver' instance.

Location

Resolver/CResolver.h

Methods

Method	Description
• GetEnumUris (see page 757)	Generates an ENUM query for the specified services and E.164 number.
• GetHostByAddress (see page 757)	Gets the hosts name using its address.
• GetHostByName (see page 757)	Gets the hosts address using its name.
• GetNameServers (see page 758)	&>
• GetNamingAuthorityPointers (see page 758)	Queries the domain for a list of naming authority pointers.
• GetServices (see page 758)	Queries hosts providing the desired service.

 SetNameServers (See page 758)

&>

Legend



2.14.1.5.1 - Methods

2.14.1.5.1.1 - CResolver::GetEnumUris Method

Generates an ENUM query for the specified services and E.164 number.

C++

```
static mxt_result GetEnumUris(IN const CString& rstrE164Number, IN const CList<CDnsPacket::SEnumServiceData>& rlstSupportedServices, OUT CList<CDnsPacket::SEnumUriData>& rlstUris, IN CString& rstrZone);
```

2.14.1.5.1.2 - CResolver::GetHostByAddress Method

Gets the hosts name using its address.

C++

```
static mxt_result GetHostByAddress(IN const CSocketAddr& rAddress, OUT CList<CString>& rlststrNames);
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddress	The address to query for names.
OUT CList<CString>& rlststrNames	The result to the query.

Returns

- resSI_TRUE: the question has been answered positively.
- resSI_FALSE: the question has been answered negatively.
- resFE_ABORT: the query timed out.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries the asynchronous resolver to find the names associated with the given address.

2.14.1.5.1.3 - CResolver::GetHostByName Method

Gets the hosts address using its name.

C++

```
static mxt_result GetHostByName(IN const CString& rstrName, OUT CList<CSocketAddr>& rlstAddresses, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrName	The name for which to find its addresses.
OUT CList<CSocketAddr>& rlstAddresses	The list of addresses.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the address to query. The default value is IPv4.

Returns

- resSI_TRUE: the question has been answered positively.
- resSI_FALSE: the question has been answered negatively.
- resFE_ABORT: the query timed out.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries the asynchronous resolver to find the addresses associated with the given name. Only addresses on the given family are returned.

2.14.1.5.1.4 - CResolver::GetNameServers Method

&>

C++

```
static void GetNameServers(OUT CList<CSocketAddr>& rlstServers);
```

2.14.1.5.1.5 - CResolver::GetNamingAuthorityPointers Method

Queries the domain for a list of naming authority pointers.

C++

```
static mxt_result GetNamingAuthorityPointers(IN const CString& rstrDomain, OUT
CList<CDnsPacket::SNaptrRecordData>& rlstPointers, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrDomain	The domain to query for naming authority pointers.
OUT CList<CDnsPacket::SNaptrRecordData>& rlstPointers	The result to the query.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the recursively resolved names.

Returns

- resSI_TRUE: the question has been answered negatively.
- resSI_FALSE: no answer has been received.
- resFE_ABORT: the query timed out.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries the asynchronous resolver to find the naming authority pointers for the given domain. Only addresses on the given family are returned.

2.14.1.5.1.6 - CResolver::GetServices Method

Queries hosts providing the desired service.

C++

```
static mxt_result GetServices(IN const CString& rstrService, OUT CList<CDnsPacket::SSrvRecordData>&
rlstServices, IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET);
```

Parameters

Parameters	Description
IN const CString& rstrService	The service to query. It needs to be in the RFC 2782 format: _._.domain.
OUT CList<CDnsPacket::SSrvRecordData>& rlstServices	The sorted list of service records for the domain.
IN CSocketAddr::EAddressFamily eFamily = CSocketAddr::eINET	The family of the recursively resolved names.

Returns

- resSI_TRUE: the question has been answered negatively.
- resSI_FALSE: no answer has been received.
- resFE_ABORT: the query timed out.
- resFE_INVALID_ARGUMENT: one or more parameters are invalid.

Description

Queries the asynchronous resolver to find the service records for the given domain. Only addresses on the given family are returned.

2.14.1.5.1.7 - CResolver::SetNameServers Method

&>

C++

```
static mxt_result SetNameServers(IN CList<CSocketAddr>& rlstServers);
```

2.14.1.6 - CResolverCache Class

This class provides an abstraction to a DNS resolver cache.

Class Hierarchy

```
CResolverCache
```

C++

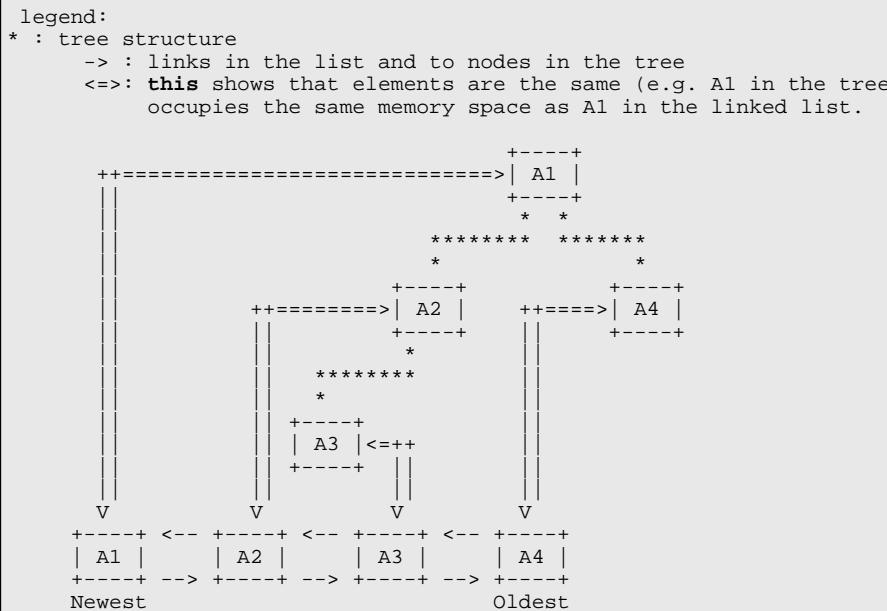
```
class CResolverCache;
```

Description

This class is a DNS resolver cache. It complies to RFCs 1035 and 2308. The cache data structure is two-fold: a tree for efficient search and an LRU list for removal of items when the cache is full. Each leaf in the tree is chained in a double-linked LRU list, in order to remove the least recently used cached answer when the cache is full. The capacity of the cache is defined by MXD_RESOLVER_CACHE_CAPACITY (see page 304).

Note that the cache cannot be instantiated by any other means than through the asynchronous resolver.

The following diagram illustrates the cache data structure:

**Location**

Resolver/CResolverCache.h

Methods

Method	Description
Compare (see page 759)	Cache sorting method.

Legend

```
Method
```

2.14.1.6.1 - Methods**2.14.1.6.1.1 - CResolverCache::Compare Method**

Cache sorting method.

C++

```
static int Compare(IN const CUncmp<CDnsPacket>& rLhs, IN const CUncmp<CDnsPacket>& rRhs, IN mxt_opaque opq);
```

Parameters

Parameters	Description
IN const CUncmp<CDnsPacket>& rLhs	The lhs of the comparison.
IN const CUncmp<CDnsPacket>& rRhs	The rhs of the comparison.
IN mxt_opaque opq	opaque parameter passed to CAA::SetComparisonFunction (see page 168).

Returns

The result from CDnsPacket::CompareQuestions.

Description

This is the sorting method used to insert items in the AA tree. It is based on the value of the question record of the packet.

2.14.1.6.2 - Friends

2.14.1.6.2.1 - friend class CAsyncResolver Friend

```
friend class CAsyncResolver;
```

2.14.1.7 - IAsyncResolverUser Class

Interface of the asynchronous resolver user.

Class Hierarchy

```
IAsyncResolverUser
```

C++

```
class IAsyncResolverUser;
```

Description

This interface needs to be implemented by users of the asynchronous resolver, in order to get the answers to their DNS requests.

Location

Resolver/IAsyncResolverUser.h

See Also

CAsyncResolver (see page 741).

Methods

Method	Description
◆ A EvAsyncResolverUserResponseReceived (see page 760)	Notification of the reception of a response to a NAPTR (ENUM) question.

Legend

◆	Method
A	abstract

2.14.1.7.1 - Methods

2.14.1.7.1.1 - EvAsyncResolverUserResponseReceived

2.14.1.7.1.1.1 - IAsyncResolverUser::EvAsyncResolverUserResponseReceived Method

Notification of the reception of a response to a NAPTR (ENUM) question.

C++

```
virtual void EvAsyncResolverUserResponseReceived(IN CList<CDnsPacket::SEnumUriData>& rlstUris, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN CList<CDnsPacket::SEnumUriData>& rlstUris	The list of ENUM URI records associated with the queried domain.

IN mxt_opaque opq	The opaque parameter passed as an argument.
-------------------	---

Description

Notifies the user that a NAPTR (ENUM) question has been answered. It may be empty, meaning the DNS question could not be answered.

2.14.1.7.1.1.2 - IAsyncResolverUser::EvAsyncResolverUserResponseReceived Method

Notification of the reception of a response to a NAPTR question.

C++

```
virtual void EvAsyncResolverUserResponseReceived(IN CList<CDnsPacket::SNaptrRecordData>& rlstPointers, IN
mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN CList<CDnsPacket::SNaptrRecordData>& rlstPointers	The list of naming authority pointer records associated with the queried domain.
IN mxt_opaque opq	The opaque parameter passed as an argument.

Description

Notifies the user that a NAPTR question has been answered. It may be empty, meaning the DNS question could not be answered.

2.14.1.7.1.1.3 - IAsyncResolverUser::EvAsyncResolverUserResponseReceived Method

Notification of the reception of a response to an SRV question.

C++

```
virtual void EvAsyncResolverUserResponseReceived(IN CList<CDnsPacket::SSrvRecordData>& rlstServices, IN
mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN CList<CDnsPacket::SSrvRecordData>& rlstServices	The list of service records associated with the queried service.
IN mxt_opaque opq	The opaque parameter passed as an argument.

Description

Notifies the user that an SRV question has been answered. It may be empty, meaning the DNS question could not be answered.

2.14.1.7.1.1.4 - IAsyncResolverUser::EvAsyncResolverUserResponseReceived Method

Notification of the reception of a response to either an A or AAAA question.

C++

```
virtual void EvAsyncResolverUserResponseReceived(IN CList<CSocketAddr>& rlstAddresses, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN CList<CSocketAddr>& rlstAddresses	The list of addresses associated with the queried name.
IN mxt_opaque opq	The opaque parameter passed as an argument.

Description

Notifies the user that either an A or AAAA question has been answered. It may contain both IPv4 and IPv6 addresses, depending on the DNS database state. It may also be empty, meaning the DNS question could not be answered.

2.14.1.7.1.1.5 - IAsyncResolverUser::EvAsyncResolverUserResponseReceived Method

Notification of the reception of a response to a PTR question.

C++

```
virtual void EvAsyncResolverUserResponseReceived(IN CList<CString>& rlststrNames, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN CList<CString>& rlststrNames	The list of names associated with the queried address.
IN mxt_opaque opq	The opaque parameter passed as an argument.

Description

Notifies the user that a PTR question has been answered. It may be empty, meaning the DNS question could not be answered.

2.14.2 - Structures

This section documents the structures of the Sources/Resolver folder.

2.15 - ServicingThread

This section documents the Sources/ServicingThread folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 762)
- Enumerations (see page 789)

2.15.1 - Classes

This section documents the classes of the Sources/ServicingThread folder.

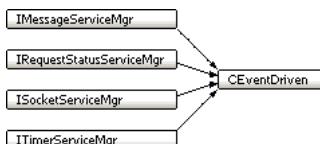
Classes

Class	Description
CEventDriven (see page 762)	Base class from which objects that use servicing threads may inherit.
CServicingThread (see page 771)	Class offering different services that are processed in the execution context of a single shared thread.
IActivationService (see page 775)	Interface providing functionalities to activate a servicing thread.
IMessageService (see page 777)	Interface providing support for posting messages to a shared message queue.
IMessageServiceMgr (see page 778)	This is the interface that must be implemented to use IMessagService (see page 777).
IRquestStatusService (see page 779)	Provides support for the detection of completion of requests through request status objects.
IRquestStatusServiceMgr (see page 781)	This is the interface that must be implemented to use IRquestStatusService (see page 779).
ISocketService (see page 782)	Provides support for the detection of events that occur on a socket.
ISocketServiceMgr (see page 784)	This is the interface that must be implemented to use ISocketService (see page 782).
ITimerService (see page 785)	Provides support for starting and stopping timers.
ITimerServiceMgr (see page 788)	This is the interface that must be implemented to use ITimerServices.

2.15.1.1 - CEventDriven Class

Base class from which objects that use servicing threads may inherit.

Class Hierarchy



C++

```
class CEventDriven : public IMessagServiceMgr, public IRquestStatusServiceMgr, public ISocketServiceMgr,
public ITimerServiceMgr;
```

Description

CEventDriven is a base class from which objects that use Servicing Threads may inherit. Its main goal is to simplify the utilization of the IMessagService (see page 777), ISocketService (see page 782) (IRquestStatusService (see page 779) under Symbian), and ITimerService (see page 785) interfaces. Please note that ISocketService (see page 782) is only available when MXD_NETWORK_ENABLE_SUPPORT (see page 298) is defined.

One area where Servicing Threads are not trivial to use is related with the releasing of an object that uses the Message Service.

Releasing is complicated by the fact that an object may be released from the same execution context as the Servicing Thread, from another one, or may also have messages pending to be processed within the queue. CEventDriven offers a simple template that may be used to get around this complexity.

Two different releasing methods exist: FinalizeAndReleaseA (see page 766) and Release (see page 768). FinalizeAndReleaseA (see page 766) should be called when messages that are pending within the queue should be processed normally before the object is deleted. Upon its return, the caller MUST NOT assume that all pending messages have been processed. Calling IsSilent from a IMessageServiceMgr (see page 778) callback always returns false for a call to FinalizeAndRelease.

Release (see page 768) should be called when messages that are pending within the queue should be silently processed before the object is deleted. Upon its return, from its point of view, the caller MAY assume that all pending messages have been processed and that the object has been released. However, from an internal point of view, it is quite possible that the object is still alive and trying to release itself.

In all cases, it is expected that an implementation calls IsSilent before any callbacks and notifications are fired.

It is possible to overload ReleaseInstance if the object must not be deleted. That may be the case if the object must wait for an asynchronous event. In such cases, the default behaviour is not advisable and must be overriden.

Warning

CEventDriven inheriting classes must be allocated on the heap.

Location

ServicingThread/CEventDriven.h

Constructors

Constructor	Description
~CEventDriven (see page 764)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CEventDriven (see page 764)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
Activate (see page 764)	Associates a Servicing Thread with this Event Driven.
DisableCompletionDetection (see page 765)	Disables the detection of request completion.
DisableEventsDetection (see page 765)	Disables the detection of events.
EnableCompletionDetection (see page 766)	Enables the detection of request completion.
EnableEventsDetection (see page 766)	Enables the detection of events.
FinalizeAndReleaseA (see page 766)	Finalizes and releases an Event Driven.
GetIComUnknown (see page 766)	Returns a pointer to the Servicing Thread. AddRef is already called.
GetServicingThread (see page 767)	Returns a pointer to the Servicing Thread. AddRef is already called.
IsCurrentExecutionContext (see page 767)	Returns whether or not the code is executing within the current execution context.
PostMessage (see page 767)	Pushes a new message into the message queue.
RegisterRequestStatus (see page 768)	Registers a request status.
RegisterSocket (see page 768)	Registers a socket.
Release (see page 768)	Releases an Event Driven.
StartTimer (see page 769)	Starts a new linear timer.
StopAllTimers (see page 770)	Stops all timers owned by a manager.
StopTimer (see page 770)	Stops a timer owned by a manager.
UnregisterRequestStatus (see page 770)	Unregisters a request status.
UnregisterSocket (see page 771)	Unregisters a socket.

ITimerServiceMgr Class

ITimerServiceMgr Class	Description
• A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

ISocketServiceMgr Class

ISocketServiceMgr Class	Description
• A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

IRequestStatusServiceMgr Class

IRequestStatusServiceMgr Class	Description
• A EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
A	abstract

2.15.1.1.1 - Constructors**2.15.1.1.1.1 - CEventDriven::CEventDriven Constructor**

Constructor.

C++

```
CEventDriven();
```

Description

Constructor.

2.15.1.1.2 - Destructors**2.15.1.1.2.1 - CEventDriven::~CEventDriven Destructor**

Destructor.

C++

```
virtual ~CEventDriven();
```

Description

Destructor.

Notes

This destructor must NEVER be explicitly called or called by the delete operator. The ONLY correct way to delete a CEventDriven (see page 762) object is via the MX_DELETE (see page 509) macro. The only reason why it is public is so that the MX_DELETE (see page 509) macro can call it.

2.15.1.1.3 - Methods**2.15.1.1.3.1 - CEventDriven::Activate Method**

Associates a Servicing Thread with this Event Driven.

C++

```
mxt_result Activate(IN IEComUnknown* pIEComUnknown, IN const char* pszName = NULL, IN uint32_t uStackSize = 0,  
IN CThread::EPriority ePriority = CThread::eNORMAL);
```

Parameters

Parameters	Description
IN IEComUnknown* pIEComUnknown	A pointer to an interface serviced by a Servicing Thread. May be NULL if a CServicingThread (see page 771) should be created using the the following parameters. When not NULL, the following parameters are ignored.
IN const char* pszName = NULL	The name associated with the spawned thread.
IN uint32_t uStackSize = 0	The stack size of the associated thread. 0 is allowed to use the default stack size.
IN CThread::EPriority ePriority = CThread::eNORMAL	The priority that must be associated with the internal thread.

Returns

resFE_INVALID_ARGUMENT resFE_INVALID_STATE

Description

Associates a Servicing Thread with this CEventDriven (see page 762) object. When pIEComUnknown is NULL, a Servicing Thread is also created.

See Also

IActivationService::Activate (see page 776)

2.15.1.1.3.2 - CEventDriven::DisableCompletionDetection Method

Disables the detection of request completion.

Disables the detection of request completion.

C++

```
mxt_result DisableCompletionDetection(IN TRequestStatus* pRequestStatus) const;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to enable completion detection.

Returns

resFE_INVALID_STATE

Description

Disables the detection of request completion.

2.15.1.1.3.3 - CEventDriven::DisableEventsDetection Method

Disables the detection of events.

C++

```
mxt_result DisableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents) const;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	The events that are to be disabled. Possible events are: uSOCKET_IN_EXCEPTION uSOCKET_READABLE uSOCKET_WRITABLE

Returns

resFE_INVALID_ARGUMENT

Description

Disables the detection of specific events for a socket. Updates the events to be detected. Only events that are true are disabled. Events that are false are not enabled.

For example, for a new socket hSocket: DisableEventDetection(hSocket, 0x06); DisableEventDetection(hSocket, 0x01); this results in all events being disabled.

See Also

ISocketService::DisableEventsDetection (see page 783)

2.15.1.1.3.4 - CEventDriven::EnableCompletionDetection Method

Enables the detection of request completion.

Enables the detection of request completion.

C++

```
mxt_result EnableCompletionDetection(IN TRequestStatus* pRequestStatus) const;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to enable completion detection.

Returns

resFE_INVALID_STATE

Description

Enables the detection of request completion.

2.15.1.1.3.5 - CEventDriven::EnableEventsDetection Method

Enables the detection of events.

C++

```
mxt_result EnableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents) const;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	The events that are to be enabled. Possible events are: uSOCKET_IN_EXCEPTION uSOCKET_READABLE uSOCKET_WRITABLE

Returns

resFE_INVALID_ARGUMENT

Description

Enables the detection of specific events for a socket. Updates the events to be detected. Only events that are true are enabled. Events that are false are not disabled.

For example, for a new socket hSocket: EnableEventDetection(hSocket, 0x06); EnableEventDetection(hSocket, 0x01); this results in all events being enabled.

See Also

ISocketService::EnableEventsDetection (See page 783)

2.15.1.1.3.6 - CEventDriven::FinalizeAndReleaseA Method

Finalizes and releases an Event Driven.

C++

```
void FinalizeAndReleaseA();
```

Description

Releases an Event Driven.

FinalizeAndReleaseA may be called from anywhere, from any thread context. It MUST NOT be called twice. Upon a FinalizeAndReleaseA method call completion, the caller MUST NOT assume that the object is released. It may still be alive and sending notification, but at one point, it will be released.

2.15.1.1.3.7 - CEventDriven::GetIEComUnknown Method Deprecated since v2.1.7 | please use GetServicingThread instead

Returns a pointer to the Servicing Thread. AddRef is already called.

C++

```
IEComUnknown* GetIEComUnknown() const;
```

Returns

A pointer to the Servicing Thread.

Description

Returns a pointer to the Servicing Thread. Make sure ReleaseSelfRef is called upon the returned pointer, since internally AddRef has already been called.

2.15.1.1.3.8 - CEventDriven::GetServicingThread Method

Returns a pointer to the Servicing Thread. AddRef is already called.

C++

```
mxt_result GetServicingThread(OUT IEComUnknown** ppEComUnknown) const;
```

Parameters

Parameters	Description
OUT IEComUnknown** ppEComUnknown	The address of a pointer to the IEComUnknown (see page 416) interface serviced by a Servicing Thread.

Returns

resFE_INVALID_ARGUMENT

Description

Returns a pointer to the Servicing Thread. Make sure ReleaseSelfRef is called upon the returned pointer, since internally AddRef has already been called.

2.15.1.1.3.9 - CEventDriven::IsCurrentExecutionContext Method

Returns whether or not the code is executing within the current execution context.

C++

```
bool IsCurrentExecutionContext() const;
```

Returns

True if it is executing within the current execution context, false if it is not.

Description

This method returns whether or not the code is executing within the current execution context.

See Also

IActivationService::IsCurrentExecutionContext (see page 777)

2.15.1.1.3.10 - CEventDriven::PostMessage Method

Pushes a new message into the message queue.

C++

```
mxt_result PostMessage(IN bool bWaitCompletion, IN unsigned int uMessage, IN TOS CMarshaler* pParameter = NULL) const;
```

Parameters

Parameters	Description
IN bool bWaitCompletion	True if the message should be processed synchronously, false if the message should be processed asynchronously.
IN unsigned int uMessage	The identifier of the message that must be posted.
IN TOS CMarshaler* pParameter = NULL	An optional CMarshaler (see page 117) parameter. The ownership of the CMarshaler (see page 117) is taken by PostMessage. Ownership is not taken if an error is returned.

Returns

resFE_INVALID_ARGUMENT

Description

PostMessage is used to push a new message onto the message queue. The message may be processed synchronously or asynchronously, depending on the value of bWaitCompletion.

The method EvMessageServiceMgrAwaken (see page 779) MUST be overloaded.

See Also

IMessageService::PostMessage (see page 778)

2.15.1.1.3.11 - CEventDriven::RegisterRequestStatus Method

Registers a request status.

Registers a request status.

C++

```
mxt_result RegisterRequestStatus(IN TRequestStatus* pRequestStatus, IN mxt_opaque opq = MX_INT32_TO_OPQ(0)) const;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be registered.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value associated with the request status.

Returns

resFE_INVALID_STATE

Description

Registers a request status. The registration associates a manager and an optional opaque value with the request status being registered.

2.15.1.1.3.12 - CEventDriven::RegisterSocket Method

Registers a socket.

C++

```
mxt_result RegisterSocket(IN mxt_hSocket hSocket, IN mxt_opaque opq = MX_INT32_TO_OPQ(0)) const;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be registered.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value associated with the socket.

Returns

resFE_INVALID_ARGUMENT

Description

Registers a socket. The registration associates a manager and an optional opaque value with the socket being registered.

Method EvSocketServiceMgrAwaken (see page 785) MUST be overloaded.

See Also

ISocketService::RegisterSocket (see page 784)

2.15.1.1.3.13 - CEventDriven::Release Method

Releases an Event Driven.

C++

```
void Release();
```

Description

Releases an Event Driven.

Release may be called from anywhere, from any thread context. It MUST NOT be called twice. Upon a Release method call completion, the caller MAY assume that the object is released. It may still be alive but acts as if it is released. The caller should not have any way of knowing that it is still alive, but at one point, it will be released.

2.15.1.1.3.14 - StartTimer

2.15.1.1.3.14.1 - CEventDriven::StartTimer Method

Starts a new linear timer.

C++

```
mxt_result StartTimer(IN unsigned int uTimer, IN uint64_t uTimeoutMs, IN mxt_opaque opq = MX_INT32_TO_OPQ(0), IN ITimerService::EPeriodicity ePeriodicity = ITimerService::ePERIODICITY_NOT_PERIODIC) const;
```

Parameters

Parameters	Description
IN unsigned int uTimer	The identifier of the timer that must be started.
IN uint64_t uTimeoutMs	The timeout before the timer elapses.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An optional opaque that may be supplied by the caller.
IN ITimerService::EPeriodicity ePeriodicity = ITimerService::ePERIODICITY_NOT_PERIODIC	The type of periodicity attached to the timer.

Returns

resFE_INVALID_ARGUMENT

Description

Starts a new timer owned by the provided manager. This may be a periodic or a one shot timer. An existing timer is first stopped as if StopTimer (see page 770) had been called before it is started again.

Once uTimeoutMs is elapsed, the manager gets notified. If the timer is periodic, it then gets automatically restarted.

The method EvTimerServiceMgrAwaken (see page 788) MUST be overloaded.

See Also

ITimerService::StartTimer (see page 786)

2.15.1.1.3.14.2 - CEventDriven::StartTimer Method

Starts a new exponential timer.

C++

```
mxt_result StartTimer(IN unsigned int uTimer, IN uint64_t uFloorTimeoutMs, IN uint64_t uCeilingTimeoutMs, IN unsigned int uMultBy, IN unsigned int uDivBy, IN bool bStopAtCeiling, IN mxt_opaque opq = MX_INT32_TO_OPQ(0), IN ITimerService::EPeriodicity ePeriodicity = ITimerService::ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST) const;
```

Parameters

Parameters	Description
IN unsigned int uTimer	The identifier of the timer that must be started.
IN uint64_t uFloorTimeoutMs	The initial timeout value when the timer is started.
IN uint64_t uCeilingTimeoutMs	The last timeout value, which will never be passed.
IN unsigned int uMultBy	The last timeout value is multiplied by uMultBy and divided by uDivBy to provide control over the variation of the timeout.
IN unsigned int uDivBy	The last timeout value is multiplied by uMultBy and divided by uDivBy to provide control over the variation of the timeout.
IN bool bStopAtCeiling	Specifies what happens when the timeout ceiling is reached. If true, the timer is stopped automatically. If false, the timer continues to fire at uCeilingTimeoutMs.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An optional opaque that may be supplied by the caller.
IN ITimerService::EPeriodicity ePeriodicity = ITimerService::ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST	The type of periodicity attached to the timer. In this case, ePERIODICITY_NOT_PERIODIC is invalid and the periodicity is related only to the inter-increment steps.

Returns

resFE_INVALID_ARGUMENT

Description

Starts a new timer owned by the provided manager. Allows the creation of exponential timers. An existing timer is first stopped as if StopTimer (see page 770) had been called before it is started again.

Control over the timeout increment is given. It is also possible to specify the behaviour when the ceiling timeout is reached.

For example, let's suppose a timer that starts at the floor timeout of 500 ms, increases exponentially by a factor of 2, and reaches its ceiling at 32000 ms is required. 500, 1000, 2000, 4000, 8000, 16000, 32000, 32000, 32000...

```
mxt_result res =
    StartTimer(pMagager, uTimer, 500, 32000, 2, 1, false);
```

See Also

ITimerService::StartTimer (see page 786)

2.15.1.1.3.15 - CEventDriven::StopAllTimers Method

Stops all timers owned by a manager.

C++

```
mxt_result StopAllTimers() const;
```

Returns

resFE_INVALID_ARGUMENT

Description

This method stops all timers that are owned by a manager.

The manager EvTimerServiceMgrAwaken (see page 788) method is called with bStopped equal to true for each stopped timer.

See Also

ITimerService::StopAllTimers (see page 787)

2.15.1.1.3.16 - CEventDriven::StopTimer Method

Stops a timer owned by a manager.

C++

```
mxt_result StopTimer(IN unsigned int uTimer) const;
```

Parameters

Parameters	Description
IN unsigned int uTimer	The timer that must be stopped.

Returns

resFE_INVALID_ARGUMENT

Description

Stops a timer. This call is simply ignored if the timer is non-existent.

The manager EvTimerServiceMgrAwaken (see page 788) method is called with bStopped equal to true if the timer exists.

See Also

ITimerService::StopTimer (see page 787)

2.15.1.1.3.17 - CEventDriven::UnregisterRequestStatus Method

Unregisters a request status.

Unregisters a request status.

C++

```
mxt_result UnregisterRequestStatus(IN TRequestStatus* pRequestStatus, OUT mxt_opaque* popq = NULL) const;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be unregistered.
OUT mxt_opaque* popq = NULL	On exit, contains the opaque value associated with the request status. It may be NULL if the value should just be discarded.

Returns

resFE_INVALID_STATE

Description

Unregisters a request status for completion detection notification.

2.15.1.1.3.18 - CEventDriven::UnregisterSocket Method

Unregisters a socket.

C++

```
mxt_result UnregisterSocket(IN mxt_hSocket hSocket, IN mxt_opaque* popq = NULL) const;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be unregistered.
IN mxt_opaque* popq = NULL	On exit, contains the opaque value associated with the socket. It MAY be NULL if the value should just be discarded.

Returns

resFE_INVALID_ARGUMENT

Description

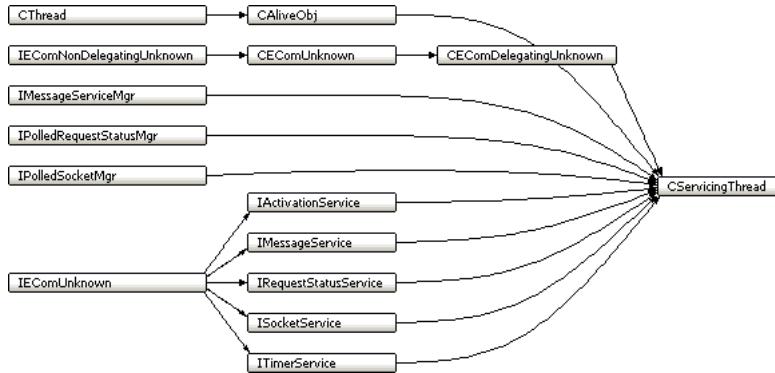
Unregisters a socket for states changes detection notification.

See Also

ISocketService::UnregisterSocket (see page 784)

2.15.1.2 - CServicingThread Class

Class offering different services that are processed in the execution context of a single shared thread.

Class Hierarchy**C++**

```
class CServicingThread : public CEventDriven, protected CAliveObj, protected IActivationService, protected IMessageService, protected IPolledRequestStatusMgr, protected IPolledSocketMgr;
```

Description

CServicingThread offers different services that are processed in the execution context of a single shared thread.

Three types of service are offered: asynchronous/synchronous messages queuing, periodic/single shot timers, and socket status change detection (request status change detection on Symbian). By using a single execution context, all events generated by these services are synchronized, helping to simplify the burden of concurrent access protection.

CServicingThread implements four interfaces: **IActivationService** (see page 775), **IMessageService** (see page 777), **ITimerService** (see page 785), and **ISocketService** (see page 782) (or **IRequestStatusService** (see page 779) on Symbian).

CServicingThread is an ECOM (see page 412) object. It must be created with a call to **CreateEComInstance** (see page 420). It is released from memory when its reference count becomes 0.

CServicingThread does the detection of inter servicing thread dead lock on synchronous calls. When a dead lock is detected, the servicing thread processes the synchronous call as if it was made from the same servicing thread; in other words, as if it was made from internally, bypassing the message queue.

```
IActivationService* pActivationService = NULL;

mxt_result res =
    CreateEComInstance(CLSID_CServicingThread,
                        NULL,
                        IID_IActivationService,
                        reinterpret_cast<void**>(&pActivationService));

if (MX_RIS_S(res))
{
    // Activate the CServicingThread. A new thread will be recreated.
    res = pActivationService->Activate();
}

if (MX_RIS_S(res))
{
    ... do something here ...
}

// Release the CServicingThread.
if (pActivationService)
{
    pActivationService->ReleaseIfRef();
    pActivationService = NULL;
}
```

Location

ServicingThread/CServicingThread.h

Constructors

Constructor	Description
CServicingThread (see page 774)	Constructor.

CAliveObj Class

CAliveObj Class	Description
CAliveObj (see page 466)	Constructor.

CEComDelegatingUnknown Class

CEComDelegatingUnknown Class	Description
CEComDelegatingUnknown (see page 414)	Constructor

CEComUnknown Class

CEComUnknown Class	Description
CEComUnknown (see page 415)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CServicingThread (see page 774)	Destructor.

CAliveObj Class

CAliveObj Class	Description
•♦ V ~CAliveObj (see page 466)	Destructor.

CEComDelegatingUnknown Class

CEComDelegatingUnknown Class	Description
•♦ V ~CEComDelegatingUnknown (see page 414)	Destructor

CEComUnknown Class

CEComUnknown Class	Description
•♦ V ~CEComUnknown (see page 416)	Destructor.

Legend

•♦	Method
V	virtual

Methods

Method	Description
•♦ CreateInstance (see page 774)	Creates a CServicingThread instance.
•♦ V InitializeInstance (see page 775)	Initializes the instance.
•♦ V NonDelegatingQueryIf (see page 775)	Queries an object for a supported interface.
•♦ V NonDelegatingReleaseSelfRef (see page 775)	Decrements the reference count on the ECOM (see page 412) implementing this interface.
•♦ V UninitializeInstance (see page 775)	Uninitializes the instance.

IPolledSocketMgr Class

IPolledSocketMgr Class	Description
•♦ A EvPolledSocketMgrEventDetected (see page 644)	Notifies the manager about newly detected events on a socket.

ISocketService Class

ISocketService Class	Description
•♦ A DisableEventsDetection (see page 783)	Disables the detection of events.
•♦ A EnableEventsDetection (see page 783)	Enables the detection of events.
•♦ A RegisterSocket (see page 784)	Registers a socket.
•♦ A UnregisterSocket (see page 784)	Unregisters a socket.

IPolledRequestStatusMgr Class

IPolledRequestStatusMgr Class	Description
•♦ A EvPolledRequestStatusMgrEventDetected (see page 643)	Notifies the manager about newly detected request completion.

IRequestStatusService Class

IRequestStatusService Class	Description
•♦ A DisableCompletionDetection (see page 780)	Disables the detection of request completion.
•♦ A EnableCompletionDetection (see page 780)	Enables the detection of request completion.
•♦ A RegisterRequestStatus (see page 781)	Registers a request status.
•♦ A UnregisterRequestStatus (see page 781)	Unregisters a request status.

ITimerService Class

ITimerService Class	Description
•♦ A StartTimer (see page 786)	Starts a new linear timer.
•♦ A StopAllTimers (see page 787)	Stops all timers owned by a manager.
•♦ A StopTimer (see page 787)	Stops a timer owned by a manager.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
•♦ A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

IMessageService Class

IMessageService Class	Description
•♦ A PostMessage (see page 778)	Pushes a new message onto the message queue.

IActivationService Class

IActivationService Class	Description
• A Activate (see page 776)	Requests the activation by using an internal thread.
• A IsCurrentExecutionContext (see page 777)	Determines if this Servicing Thread is the current execution context.

CAliveObj Class

CAliveObj Class	Description
• A Activate (see page 466)	Gives life to the object (Start the associated thread).
• A IsAlive (see page 467)	Indicates the state of the object (thread-safe).
• A IsDead (see page 467)	Indicates the state of the object (thread-safe).
• A IsUnborn (see page 467)	Indicates the state of the object (thread-safe).

CEComUnknown Class

CEComUnknown Class	Description
• V InitializeInstance (see page 416)	Initializes the instance.

Legend

•	Method
V	virtual
A	abstract

2.15.1.2.1 - Constructors**2.15.1.2.1.1 - CServicingThread::CServicingThread Constructor**

Constructor.

C++

```
CServicingThread( IN IEComUnknown* pOuterIEComUnknown = NULL );
```

Parameters

Parameters	Description
IN IEComUnknown* pOuterIEComUnknown = NULL	Pointer to a IEComUnknown (see page 416) interface.

Description

Constructor.

2.15.1.2.2 - Destructors**2.15.1.2.2.1 - CServicingThread::~CServicingThread Destructor**

Destructor.

C++

```
virtual ~CServicingThread();
```

Description

Destructor.

2.15.1.2.3 - Methods**2.15.1.2.3.1 - CServicingThread::CreateInstance Method**

Creates a CServicingThread (see page 771) instance.

C++

```
static mxxt_result CreateInstance(IN IEComUnknown* pOuterIEComUnknown, OUT CEComUnknown** ppCEComUnknown);
```

Parameters

Parameters	Description
IN IEComUnknown* pOuterIEComUnknown	Pointer to a IEComUnknown (see page 416) interface.
OUT CEComUnknown** ppCEComUnknown	Pointer to the created CServicingThread (see page 771) instance.

Description

This method creates a CServicingThread (see page 771) using the ECom mechanism.

2.15.1.2.3.2 - CServicingThread::InitializeInstance Method

Initializes the instance.

C++

```
virtual MX_DECLARE_DELEGATING_IECOMUNKNOWN mxt_result InitializeInstance();
```

Returns

A mxt_result error code. This depends on the implementation of the initialize instance.

Description

Initializes information inside the ECOM instance.

See Also

UninitializeInstance, Reference counting rules (see page 416)

2.15.1.2.3.3 - CServicingThread::NonDelegatingQueryIf Method

Queries an object for a supported interface.

C++

```
virtual mxt_result NonDelegatingQueryIf(IN mxt_iid iidRequested, OUT void** ppInterface);
```

2.15.1.2.3.4 - CServicingThread::NonDelegatingReleaseIfRef Method

Decrement the reference count on the ECOM (see page 412) implementing this interface.

C++

```
virtual unsigned int NonDelegatingReleaseIfRef();
```

2.15.1.2.3.5 - CServicingThread::UninitializeInstance Method

Uninitializes the instance.

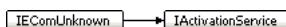
C++

```
virtual void UninitializeInstance(OUT bool* pbDeleteThis);
```

2.15.1.3 - IActivationService Class

Interface providing functionalities to activate a servicing thread.

Class Hierarchy



C++

```
class IActivationService : public IEComUnknown;
```

Description

The interface iidActivationService (IActivationService) provides functionalities to activate a Servicing Thread. The activation gives life to the Servicing Thread, giving it an execution context to process its services. There are two different activation mechanisms: the first is the activation throughout the creation of an internal thread that provides the execution context for the Servicing Thread, the second is a periodic polling of the activation method from an execution context, thus providing an external execution context to the Servicing Thread. This second mode is most useful when an application already has its own execution context. Instead of forcing the creation of another thread, the framework allows the activation of the Servicing Thread from an application thread.

Location

ServicingThread/IActivationService.h

See Also

IMessageService (see page 777), ISocketService (see page 782), ITimerService (see page 785)

Methods

Method	Description
• A Activate (see page 776)	Requests the activation by using an internal thread.
• A IsCurrentExecutionContext (see page 777)	Determines if this Servicing Thread is the current execution context.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
A	abstract

2.15.1.3.1 - Methods**2.15.1.3.1.1 - Activate****2.15.1.3.1.1.1 - IActivationService::Activate Method**

Requests the activation by using an internal thread.

C++

```
virtual mxt_result Activate(IN const char* pszName = NULL, IN uint32_t uStackSize = 0, IN CThread::EPriority ePriority = CThread::eNORMAL) = 0;
```

Parameters

Parameters	Description
IN const char* pszName = NULL	The name associated with the spawned thread.
IN uint32_t uStackSize = 0	The stack size of the associated thread. 0 is allowed to use the default stack size.
IN CThread::EPriority ePriority = CThread::eNORMAL	The priority that must be associated with the internal thread.

Returns

resFE_INVALID_ARGUMENT

Description

This method should be called when an internal thread should be created and assigned to supply the execution context.

2.15.1.3.1.1.2 - IActivationService::Activate Method

Requests the activation by using an external thread.

C++

```
virtual mxt_result Activate(IN uint64_t uTimeoutMs, OUT bool* pbReadyToRelease) = 0;
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs	The maximum period of time that may be passed to process the execution context.
OUT bool* pbReadyToRelease	On exit, contains true if the execution context is no longer required and may be released.

Returns

```
resFE_INVALID_ARGUMENT resFE_INVALID_STATE
```

Description

This method should be called periodically when an external thread is used to supply the execution context.

The fact that a reference is required to call this method renders impossible the automatic release of the object. To resolve this issue, the caller may pass a pointer to a bool that is set to true if the only reference left is this one. Normally, the caller should then release the reference it owns and the execution context would then be released.

Warning

This method MUST ALWAYS be called at least once before any other method is called on a Servicing Thread. A dead lock will occur if this is not enforced.

This method MUST ALWAYS be called from the same thread. Trying to call this method from multiple threads, even if synchronized, leads to undefined behaviour. This is because each thread has its own Thread Specific Data ("TSD"). Code that uses the TSD does not behave correctly under these circumstances.

2.15.1.3.1.2 - IActivationService::IsCurrentExecutionContext Method

Determines if this Servicing Thread is the current execution context.

C++

```
virtual bool IsCurrentExecutionContext() = 0;
```

Returns

true if this Servicing Thread is the current execution context, false otherwise.

Description

Determines if this Servicing Thread is the current execution context.

2.15.1.4 - IMessageService Class

Interface providing support for posting messages to a shared message queue.

Class Hierarchy**C++**

```
class IMessageService : public IEComUnknown;
```

Description

The message queuing service provides support for posting messages to a shared message queue. The message is then processed within the execution context. It eventually generates an event notification to the associated manager. When a message is posted within the message queue, it is possible for the user to specify whether the message should be processed synchronously or asynchronously. The user is also allowed to attach an optional CMarshaler (see page 117) reference to the message it is posting. The reference is given back to the manager upon notification. The CMarshaler (see page 117) is used to store additional parameters that are required to process the message.

Message queuing works differently for messages that must be processed synchronously or asynchronously depending on the execution context from which the message is posted. For asynchronous messages, the message is always added to the message queue. For synchronous messages, if the posting execution context is different than the processing execution context, the message is added to the shared message queue and the posting execution context is put to sleep until the message is processed. If the message posting execution context is the same as the processing execution context, the associated manager is called directly, bypassing the message queue. The reason is simple: if this was not the case, the code would deadlock because the message would never be processed.

The interface iidMessageService (IMessageService) provides the method PostMessage (see page 778) to request the processing of a message. The bWaitCompletion parameter controls how the message is processed: asynchronously or synchronously. The uMessage parameter identifies the message.

Notifications are sent to the manager through the method IMessageServiceMgr::EvMessageServiceMgrAwaken (see page 779). The manager must overload this method to receive notifications. The bWaitingCompleting parameter of the manager notification EvMessageServiceMgr method is true if the poster is waiting for the message processing completion.

Location

ServicingThread/IMessageService.h

See Also

IActivationService (see page 775), IMessageServiceMgr (see page 778), ISocketService (see page 782), ITimerService (see page 785)

Methods

Method	Description
• A PostMessage (see page 778)	Pushes a new message onto the message queue.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.15.1.4.1 - Methods

2.15.1.4.1.1 - IMessageService::PostMessage Method

Pushes a new message onto the message queue.

C++

```
virtual mxt_result PostMessage(IN IMessageServiceMgr* pManager, IN bool bWaitCompletion, IN unsigned int uMessageId, IN TOS CMarshaler* pParameter = NULL) = 0;
```

Parameters

Parameters	Description
IN IMessageServiceMgr* pManager	The manager to be notified.
IN bool bWaitCompletion	True if the message should be processed synchronously, false if the message should be processed asynchronously.
IN unsigned int uMessageId	The identifier of the message that must be posted.
IN TOS CMarshaler* pParameter = NULL	An optional CMarshaler (see page 117) parameter. The ownership of the CMarshaler (see page 117) is taken by PostMessage. Ownership is not taken if an error is returned.

Returns

resFE_INVALID_ARGUMENT

Description

PostMessage is used to push a new message onto the message queue. The message may be processed synchronously or asynchronously, depending on the value of bWaitCompletion.

2.15.1.5 - IMessageServiceMgr Class

This is the interface that must be implemented to use IMessageService (see page 777).

Class Hierarchy

IMessageServiceMgr

C++

```
class IMessageServiceMgr;
```

Description

This is the interface that must be implemented to use IMessageService (see page 777).

Location

ServicingThread/IMessageServiceMgr.h

See Also

IActivationService (see page 775), IMessageServiceMgr, ISocketService (see page 782), ITimerService (see page 785)

Methods

Method	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
A	abstract

2.15.1.5.1 - Methods**2.15.1.5.1.1 - IMessageServiceMgr::EvMessageServiceMgrAwaken Method**

Notifies the manager that a new message must be processed.

C++

```
virtual void EvMessageServiceMgrAwaken( IN bool bWaitingCompletion, IN unsigned int uMessageId, IN CMarshaler* pParameter ) = 0;
```

Parameters

Parameters	Description
IN bool bWaitingCompletion	True if the message is being processed synchronously, false if processed asynchronously.
IN unsigned int uMessageId	The identifier of the message that must be posted.
IN CMarshaler* pParameter	An optional CMarshaler (see page 117) parameter.

Description

Notifies the manager that a new message must be processed.

2.15.1.6 - IRequestStatusService Class Symbian OS ONLY

Provides support for the detection of completion of requests through request status objects.

Class Hierarchy**C++**

```
class IRequestStatusService : public IEComUnknown;
```

Description

The request status service provides support for the detection of request status event completion.

Before receiving notifications about the completion of a request status, the request status must first be registered. The registration associates a manager and an optional opaque value with the request status being registered.

From now on, completion detection may be enabled. It is also possible to disable the completion detection.

The associated request status manager is notified when an enabled request status is in the completed state. Following notification, the detected request status is automatically disabled.

The interface iidRequestStatusService (IRequestStatusService) provides methods to register and unregister request status and methods to enable and disable completion detection on registered request status.

Notifications are sent to the manager through the method IRequestStatusServiceMgr::EvRequestStatusServiceMgrAwaken (see page 782). The manager must overload this method to receive notifications.

Notes

This interface is only available on Symbian. Please see ISocketService (see page 782), which must be used on other OSes.

Location

ServicingThread/IRequestStatusService.h

See Also

IActivationService (see page 775), IMessageService (see page 777), IRequestStatusServiceMgr (see page 781), ITimerService (see page 785)

Methods

Method	Description
• A DisableCompletionDetection (see page 780)	Disables the detection of request completion.
• A EnableCompletionDetection (see page 780)	Enables the detection of request completion.
• A RegisterRequestStatus (see page 781)	Registers a request status.
• A UnregisterRequestStatus (see page 781)	Unregisters a request status.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
A	abstract

2.15.1.6.1 - Methods**2.15.1.6.1.1 - IRequestStatusService::DisableCompletionDetection Method**

Disables the detection of request completion.

C++

```
virtual mxt_result DisableCompletionDetection( IN TRequestStatus* pRequestStatus ) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to enable completion detection.

Returns

resFE_INVALID_ARGUMENT

Description

Disables the detection of request completion.

2.15.1.6.1.2 - IRequestStatusService::EnableCompletionDetection Method

Enables the detection of request completion.

C++

```
virtual mxt_result EnableCompletionDetection( IN TRequestStatus* pRequestStatus ) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status for which to enable completion detection.

Returns

resFE_INVALID_ARGUMENT

Description

Enables the detection of request completion.

2.15.1.6.1.3 - IRequestStatusService::RegisterRequestStatus Method

Registers a request status.

C++

```
virtual mxt_result RegisterRequestStatus(IN TRequestStatus* pRequestStatus, IN IRequestStatusServiceMgr* pManager, IN mxt_opaque opq = MX_INT32_TO_OPQ(0)) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be registered.
IN IRequestStatusServiceMgr* pManager	The manager to be notified.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value associated with the request status.

Returns

resFE_INVALID_ARGUMENT

Description

Registers a request status. The registration associates a manager and an optional opaque value with the request status being registered.

2.15.1.6.1.4 - IRequestStatusService::UnregisterRequestStatus Method

Unregisters a request status.

C++

```
virtual mxt_result UnregisterRequestStatus(IN TRequestStatus* pRequestStatus, OUT mxt_opaque* popq = NULL) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The request status that must be unregistered.
OUT mxt_opaque* popq = NULL	On exit, contains the opaque value associated with the request status. It may be NULL if the value should just be discarded.

Returns

resFE_INVALID_ARGUMENT

Description

Unregisters a request status for completion detection notification.

2.15.1.7 - IRequestStatusServiceMgr Class Symbian OS ONLY

This is the interface that must be implemented to use IRequestStatusService (see page 779).

Class Hierarchy

```
IRequestStatusServiceMgr
```

C++

```
class IRequestStatusServiceMgr;
```

Description

This is the interface that must be implemented to use IRequestStatusService (see page 779).

Location

ServicingThread/IRequestStatusServiceMgr.h

See Also

IRequestStatusService (see page 779)

Methods

Method	Description
EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

Legend

	Method
	abstract

2.15.1.7.1 - Methods**2.15.1.7.1.1 - IRequestStatusServiceMgr::EvRequestStatusServiceMgrAwaken Method**

Notifies the manager about newly completed request.

C++

```
virtual void EvRequestStatusServiceMgrAwaken(IN TRequestStatus* pRequestStatus, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN TRequestStatus* pRequestStatus	The identifier of the request status.
IN mxt_opaque opq	The opaque value provided at registration.

Description

The request status is registered and the detection is enabled. The request completed and its status has been updated. From now on, completion on this specific request will no longer be detected. The detection needs to be re-enabled before the completion may be detected again.

For example: A request status is registered and detection is enabled. At one point, the manager gets notified about the completion. Following this notification, the manager is never notified again about the completion of any new request issued reusing this request status.

2.15.1.8 - ISocketService Class

Provides support for the detection of events that occur on a socket.

Class Hierarchy**C++**

```
class ISocketService : public IEComUnknown;
```

Description

The socket service provides support for the detection of events that occur on a socket. Three different types of events may be detected: socket readability, socket writability, and socket exceptions.

Before receiving notifications about the detection of one or more events that occur on a socket, the socket must first be registered. The registration associates a manager and an optional opaque value with the socket being registered.

From now on, events to be detected may then be enabled. It is also possible to disable the detection of events.

The associated socket manager is notified when enabled events are detected. Following notification, the detected events are automatically disabled.

For example, suppose that readability and writability events detection is enabled. Upon socket readability detection, the readability event detection gets disabled. The writability event detection is still enabled because it has not been detected.

The interface iidSocketService (ISocketService) provides methods to register and unregister sockets and methods to enable and disable events detection on registered sockets. The uEvents parameter is a bitfield that specifies which events must be enabled or disabled.

Notifications are sent to the manager through the method ISocketServiceMgr::EvSocketServiceMgrAwaken (see page 785). The manager must overload this method to receive notifications.

Notes

This interface is not available on Symbian. Please see IRequestStatusService (see page 779), which must be used instead.

Location

ServicingThread/ISocketService.h

See Also

[IActivationService](#) (see page 775), [IMessageService](#) (see page 777), [ISocketServiceMgr](#) (see page 784), [ITimerService](#) (see page 785)

Methods

Method	Description
• DisableEventsDetection (see page 783)	Disables the detection of events.
• EnableEventsDetection (see page 783)	Enables the detection of events.
• RegisterSocket (see page 784)	Registers a socket.
• UnregisterSocket (see page 784)	Unregisters a socket.

IEComUnknown Class

IEComUnknown Class	Description
• AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• QueryIf (see page 418)	Queries an object for a supported interface.
• ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.15.1.8.1 - Methods**2.15.1.8.1.1 - [ISocketService::DisableEventsDetection](#) Method**

Disables the detection of events.

C++

```
virtual mxt_result DisableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents) = 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	The events that are to be disabled. Possible events are: uSOCKET_IN_EXCEPTION uSOCKET_READABLE uSOCKET_WRITABLE

Returns

[resFE_INVALID_ARGUMENT](#)

Description

Disables the detection of specific events for a socket. Updates the events to be detected. Only events that are true are disabled. Events that are false are not enabled.

For example, with a new socket hSocket `DisableEventDetection(hSocket, 0x06); DisableEventDetection(hSocket, 0x01);` this results in all events being disabled.

2.15.1.8.1.2 - [ISocketService::EnableEventsDetection](#) Method

Enables the detection of events.

C++

```
virtual mxt_result EnableEventsDetection(IN mxt_hSocket hSocket, IN unsigned int uEvents) = 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	The events that are to be enabled. Possible events are: uSOCKET_IN_EXCEPTION uSOCKET_READABLE uSOCKET_WRITABLE

Returns

resFE_INVALID_ARGUMENT

Description

Enables the detection of specific events for a socket. Updates the events to be detected. Only events that are true are enabled. Events that are false are not disabled.

For example, with a new socket hSocket EnableEventDetection(hSocket, 0x06); EnableEventDetection(hSocket, 0x01); this results in all events being enabled.

2.15.1.8.1.3 - ISocketService::RegisterSocket Method

Registers a socket.

C++

```
virtual mxt_result RegisterSocket(IN mxt_hSocket hSocket, IN ISocketServiceMgr* pManager, IN mxt_opaque opq = MX_INT32_TO_OPQ(0)) = 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be registered.
IN ISocketServiceMgr* pManager	The manager to be notified.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An opaque value associated with the socket.

Returns

resFE_INVALID_ARGUMENT

Description

Registers a socket. The registration associates a manager and an optional opaque value with the socket being registered.

2.15.1.8.1.4 - ISocketService::UnregisterSocket Method

Unregisters a socket.

C++

```
virtual mxt_result UnregisterSocket(IN mxt_hSocket hSocket, OUT mxt_opaque* popq = NULL) = 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket that must be unregistered.
OUT mxt_opaque* popq = NULL	On exit, contains the opaque value associated with the socket. It may be NULL if the value should just be discarded.

Returns

resFE_INVALID_ARGUMENT

Description

Unregisters a socket for states changes detection notification.

2.15.1.9 - ISocketServiceMgr Class

This is the interface that must be implemented to use ISocketService (see page 782).

Class Hierarchy

```
ISocketServiceMgr
```

C++

```
class ISocketServiceMgr;
```

Description

This is the interface that must be implemented to use ISocketService (see page 782).

Location

ServicingThread/ISocketServiceMgr.h

See Also

ISocketService (see page 782)

Methods

Method	Description
● A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

Legend

●	Method
A	abstract

2.15.1.9.1 - Methods**2.15.1.9.1.1 - ISocketServiceMgr::EvSocketServiceMgrAwaken Method**

Notifies the manager about newly detected events on a socket.

C++

```
virtual void EvSocketServiceMgrAwaken(IN mxt_hSocket hSocket, IN unsigned int uEvents, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_hSocket hSocket	The identifier of the socket.
IN unsigned int uEvents	The events detected. Possible events are: uSOCKET_IN_EXCEPTION uSOCKET_READABLE uSOCKET_WRITABLE
IN mxt_opaque opq	The opaque value provided at registration.

Description

The socket is registered and the detection of one or more events is enabled. At least one of these events has just been detected. From now on, these specific events will no longer be detected. They need to be re-enabled before they may be detected again.

For example: A socket is registered for readability and writability detection. At one point, the manager gets notified about the socket readability. Following this notification, the manager is never notified again about the readability of the socket. It may still however be notified of the writability of the socket.

2.15.1.10 - ITimerService Class

Provides support for starting and stopping timers.

Class Hierarchy**C++**

```
class ITimerService : public IEComUnknown;
```

Description

The timer service provides support for starting and stopping timers. Two different types of timers currently exist: linear and exponential.

A linear timer always fires after a fixed timeout. A linear timer may be single shot or multiple shot, depending on its associated periodicity type.

An exponential timer fires with an ever increasing timeout controlled by a configured (uMultBy, uDivBy) ratio. It first starts from a configured floor timeout and exponentially increases up to a configured ceiling timeout. A configured bStopAtCeilingOnce determines what happens when the ceiling timeout is reached. If true, the timer is stopped. If false, it continues to be fired but with a constant timeout. This constant timeout is equal to the last timeout calculated (exponentially) prior to reaching the ceiling.

An optional opaque may be configured when the StartTimer (see page 786) is called. This opaque is sent back to the user through the manager notification. The user is free to use whatever value. The service does not try to interpret it.

Stopping a timer triggers a notification to the manager, indicating that the timer has been stopped. A notification is also triggered when a timer elapsed.

Notifications are sent to the manager through the method `ITimerServiceMgr::EvTimerServiceMgrAwaken` (see page 788). The manager must overload this method to receive notifications.

Location

`ServicingThread/ITimerService.h`

See Also

`IActivationService` (see page 775), `IMessageService` (see page 777), `ISocketService` (see page 782)

Methods

Method	Description
• A <code>StartTimer</code> (see page 786)	Starts a new linear timer.
• A <code>StopAllTimers</code> (see page 787)	Stops all timers owned by a manager.
• A <code>StopTimer</code> (see page 787)	Stops a timer owned by a manager.

IEComUnknown Class

IEComUnknown Class	Description
• A <code>AddIfRef</code> (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A <code>QueryIf</code> (see page 418)	Queries an object for a supported interface.
• A <code>ReleaseIfRef</code> (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

•	Method
A	abstract

2.15.1.10.1 - Methods

2.15.1.10.1.1 - StartTimer

2.15.1.10.1.1.1 - `ITimerService::StartTimer` Method

Starts a new linear timer.

C++

```
virtual mxt_result StartTimer(IN ITimerServiceMgr* pManager, IN unsigned int uTimerId, IN uint64_t uTimeoutMs,
IN mxt_opaque opq = MX_INT32_TO_OPQ(0), IN EPeriodicity ePeriodicity = ePERIODICITY_NOT_PERIODIC) = 0;
```

Parameters

Parameters	Description
IN ITimerServiceMgr* pManager	The manager that owns the timer.
IN unsigned int uTimerId	The identifier of the timer that must be started.
IN uint64_t uTimeoutMs	The timeout before the timer elapses.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An optional opaque that may be supplied by the caller.
IN EPeriodicity ePeriodicity = ePERIODICITY_NOT_PERIODIC	The type of periodicity attached to the timer.

Returns

`resFE_INVALID_ARGUMENT`

Description

Starts a new timer that is owned by the provided manager. This may be a periodic or a one shot timer. An existing timer is first stopped as if `StopTimer` (see page 787) had been called before it is started again.

Once `uTimeoutMs` is elapsed, the manager gets notified. If the timer is periodic, it then gets automatically restarted.

2.15.1.10.1.1.2 - `ITimerService::StartTimer` Method

Starts a new exponential timer.

C++

```
virtual mxt_result StartTimer(IN ITimerServiceMgr* pManager, IN unsigned int uTimerId, IN uint64_t
```

```
uFloorTimeoutMs, IN uint64_t uCeilingTimeoutMs, IN unsigned int uMultBy, IN unsigned int uDivBy, IN bool
bStopAtCeiling, IN mxt_opaque opq = MX_INT32_TO_OPQ(0), IN EPeriodicity ePeriodicity =
ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST) = 0;
```

Parameters

Parameters	Description
IN ITimerServiceMgr* pManager	The manager that owns the timer.
IN unsigned int uTimerId	The identifier of the timer that must be started.
IN uint64_t uFloorTimeoutMs	The initial timeout value when the timer is started.
IN uint64_t uCeilingTimeoutMs	The last timeout value, which will never be passed.
IN unsigned int uMultBy	The last timeout value is multiplied by uMultBy and divided by uDivBy to provide control over the variation of the timeout.
IN unsigned int uDivBy	The last timeout value is multiplied by uMultBy and divided by uDivBy to provide control over the variation of the timeout.
IN bool bStopAtCeiling	Specifies what happens when the timeout ceiling is reached. If true, the timer is stopped automatically. If false, the timer continues to fire at uCeilingTimeoutMs.
IN mxt_opaque opq = MX_INT32_TO_OPQ(0)	An optional opaque that may be supplied by the caller.
IN EPeriodicity ePeriodicity = ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST	The type of periodicity attached to the timer. In this case, ePERIODICITY_NOT_PERIODIC is invalid and the periodicity is related only to the inter-increment steps.

Returns

resFE_INVALID_ARGUMENT

Description

Starts a new timer that is owned by the provided manager. Allows the creation of exponential timers. An existing timer is first stopped as if StopTimer (see page 787) had been called before it is started again.

Control over the timeout increment is given. It is also possible to specify the behaviour when the ceiling timeout is reached.

For example, suppose that a timer that starts at the floor timeout of 500 ms, increases exponentially by a factor of 2, and reaches its ceiling at 32000 ms is required. 500, 1000, 2000, 4000, 8000, 16000, 32000, 32000, 32000...

```
mxt_result res =
StartTimer(pManager, uTimer, 500, 32000, 2, 1, false);
```

2.15.1.10.1.2 - ITimerService::StopAllTimers Method

Stops all timers owned by a manager.

C++

```
virtual mxt_result StopAllTimers(IN ITimerServiceMgr* pManager) = 0;
```

Parameters

Parameters	Description
IN ITimerServiceMgr* pManager	The manager that owns the timers.

Returns

resFE_INVALID_ARGUMENT

Description

This method stops all timers that are owned by a manager. The manager EvTimerServiceMgrAwaken method is called with bStopped equal to true for each stopped timer.

2.15.1.10.1.3 - ITimerService::StopTimer Method

Stops a timer owned by a manager.

C++

```
virtual mxt_result StopTimer(IN ITimerServiceMgr* pManager, IN unsigned int uTimerId) = 0;
```

Parameters

Parameters	Description
IN ITimerServiceMgr* pManager	The manager that owns the timers.
IN unsigned int uTimerId	The timer that must be stopped.

Returns

resFE_INVALID_ARGUMENT

Description

Stops a timer owned by a manager. This call is simply ignored if the timer is inexistent.

The manager EvTimerServiceMgrAwaken method is called with bStopped equal to true if the timer exists.

2.15.1.11 - ITimerServiceMgr Class

This is the interface that must be implemented to use ITimerServices.

Class Hierarchy

C++

```
class ITimerServiceMgr;
```

Description

This is the interface that must be implemented to use ITimerServices.

Location

ServicingThread/ITimerServiceMgr.h

See Also

IActivationService (see page 775), IMessageService (see page 777), ISocketService (see page 782), ITimerServiceMgr

Methods

Method	Description
◆ A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

Legend

	Method
	abstract

2.15.1.11.1 - Methods**2.15.1.11.1.1 - ITimerServiceMgr::EvTimerServiceMgrAwaken Method**

Notifies the manager that a new timer elapsed or has been stopped.

C++

```
virtual void EvTimerServiceMgrAwaken(IN bool bStopped, IN unsigned int uTimerId, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN bool bStopped	True if the timer has been explicitly stopped, false if it normally elapsed. For an exponential timer, true is also sent if the ceiling has been reached and bStopAtCeiling is true.
IN unsigned int uTimerId	The identifier of the timer.
IN mxt_opaque opq	An optional opaque supplied by the caller.

Description

Notifies the manager that a new timer elapsed.

For non-periodic timers, this method is called once, with bStopped equal to false. For example, EvTimerServiceMgrAwaken(false, uTimerId, opq)

For periodic and exponential timers, this method is called as many times as the timer elapses + 1. The + 1 means that the timer has been destroyed. For example, EvTimerServiceMgrAwaken(false, uTimerId, opq) ... EvTimerServiceMgrAwaken(false, uTimerId, opq) always followed by EvTimerServiceMgrAwaken(true, uTimerId, opq)

If a timer is stopped with StopTimer or StopAllTimers, this method is also called with bStopped set to true.

2.15.2 - Enumerations

This section documents the enumerations of the Sources/ServicingThread folder.

Enumerations

Enumeration	Description
EPeriodicity (see page 789)	Defines the possible periodicity of a timer.

2.15.2.1 - ITimerService::EPeriodicity Enumeration

Defines the possible periodicity of a timer.

C++

```
enum EPeriodicity {
    ePERIODICITY_NOT_PERIODIC,
    ePERIODICITY_REAJUST_WITH_CURRENT_TIME,
    ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_CYCLE_LOST,
    ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST
};
```

Description

Defines the possible periodicity types that a timer may have.

Members

Members	Description
ePERIODICITY_NOT_PERIODIC	The timer is not periodic. Not available to exponential timers.
ePERIODICITY_REAJUST_WITH_CURRENT_TIME	The next notification will occur at (system timer plus timeout).
ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_CYCLE_LOST	The next notification will occur at (previous notification plus timeout). If the next computed notification is already in the past, timeout is added once or more until the computed notification is in the future. For example: suppose that a timer was started at current time equal to 0ms with a timeout equal to 10ms. Suppose also that the previous notification occurred at 30ms and the current time is now 53ms, then the next computed notification will be equal to 60ms. Thus, the timer will sleep for 7ms.
ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST	The next notification will occur at (previous notification plus timeout). For example: suppose that a timer was started at current time equal to 0ms with a timeout equal to 10ms. Suppose also that the previous notification occurred at 30ms and the current time is now 53ms, then the notification will be done twice sequentially, once for 40ms and another for 50ms. The next computed notification will be equal to 60ms. Thus, the timer will sleep for 7ms.

2.16 - Startup

This section documents the Sources/Startup folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 789)
- Structures (see page 791)
- Variables (see page 794)

2.16.1 - Classes

This section documents the classes of the Sources/Startup folder.

Classes

Class	Description
CFrameworkInitializer (see page 789)	This class is responsible of doing the initialization and finalization of the Framework.

2.16.1.1 - CFrameworkInitializer Class

This class is responsible of doing the initialization and finalization of the Framework.

Class Hierarchy

CFrameworkInitializer

C++

```
class CFrameworkInitializer;
```

Description

CFrameworkInitializer::Initialize (see page 791) must be called before any of the functionality from the Framework is used. CFrameworkInitializer::Finalize (see page 790) must be called when the functionality from Framework is no longer needed.

This class is not thread safe. If more than one call to Initialize (see page 791) or Finalize (see page 790) are necessary, these calls must come from the same thread or be protected against concurrent access. For example, if the M5T RTP and M5T SIP are being used, since each needs to make a separate call to CFrameworkInitializer Initialize (see page 791) and Finalize (see page 790), these calls must be thread safe.

Methods

Method	Description
Finalize (see page 790)	Finalizes the Framework.
GetInitParameters (see page 790)	This is GetInitParameters, a member of the class CFrameworkInitializer.
GetInitParametersCount (see page 791)	This is GetInitParametersCount, a member of the class CFrameworkInitializer.
Initialize (see page 791)	Initializes the Framework.
InitializeSymbianNetwork (see page 791)	Initializes Symbian 9.1 network functionality.
IsInitialized (see page 791)	Gets the initialization status of the Framework.

Legend

	Method
--	--------

2.16.1.1.1 - Methods

2.16.1.1.1.1 - Finalize

2.16.1.1.1.1.1 - CFrameworkInitializer::Finalize Method

Finalizes the Framework.

C++

```
static void Finalize();
```

Description

This method finalizes the Framework. It must be called when the functionality from the Framework is no longer needed.

2.16.1.1.1.1.2 - CFrameworkInitializer::Finalize Method

Finalizes the Framework and returns information about the state of the framework at the point where the finalize method was invoked.

C++

```
static unsigned int Finalize(OUT SFrameworkFinalizeInfo* pstFinalizeInfo);
```

Parameters

Parameters	Description
OUT SFrameworkFinalizeInfo* pstFinalizeInfo	Pointer to the structure used to report the information about the final state of the Framework. If this parameter is not NULL and the initialization count returned is greater than 0, the method will always report memory leaks.

Returns

The current initialization count as an unsigned integer.

Description

This method finalizes the Framework and returns information about the state of the framework at the point where the Finalize method was invoked. This method must be called when the functionalities from the Framework are no longer needed.

2.16.1.1.1.2 - CFrameworkInitializer::GetInitParameters Method

This is GetInitParameters, a member of the class CFrameworkInitializer.

C++

```
static const char* const* GetInitParameters();
```

2.16.1.1.1.3 - CFrameworkInitializer::GetInitParametersCount Method

This is GetInitParametersCount, a member of the class CFrameworkInitializer.

C++

```
static unsigned int GetInitParametersCount();
```

2.16.1.1.1.4 - Initialize

2.16.1.1.1.4.1 - CFrameworkInitializer::Initialize Method

Initializes the Framework.

C++

```
static mxt_result Initialize();
static mxt_result Initialize(IN const char* const* const ppszInitParameters, IN unsigned int uNumInitParameters);
```

Returns

- resS_OK
- Other(s): A result code that can be returned by the Initialize methods.

Description

This method does the initialization of the Framework. It MUST be called before any of the functionality of the Framework is used.

2.16.1.1.1.5 - CFrameworkInitializer::InitializeSymbianNetwork Method

Initializes Symbian 9.1 network functionality.

C++

```
static void InitializeSymbianNetwork(RConnection* pConnection, RSocketServ* pSocketServer);
```

Parameters

Parameters	Description
RConnection* pConnection	Pointer to the RConnection object to be used by the Framework.
RSocketServ* pSocketServer	Pointer to the RSocketServ object to be used by the Framework.

Description

Initializes Symbian 9.1 network by setting a RConnection and RSocketServ object to be used by all sockets created by the Framework. Please refer to the Symbian SDK documentation for more information on the RConnection and RSocketServ objects.

2.16.1.1.1.6 - CFrameworkInitializer::IsInitialized Method

Gets the initialization status of the Framework.

C++

```
static bool IsInitialized();
```

Returns

True if the Framework is initialized, false otherwise.

Description

This method gets the initialization status of the Framework.

2.16.2 - Structures

This section documents the structures of the Sources/Startup folder.

Structs

Struct	Description
SFrameworkFinalizeInfo (See page 792)	Structure used to report information at finalization time.
SMemoryInfo (See page 793)	Holds a snapshot of memory at finalization time.

2.16.2.1 - CFrameworkInitializer::SFrameworkFinalizeInfo Struct

Structure used to report information at finalization time.

Class Hierarchy

C++

```
struct SFrameworkFinalizeInfo : public CMemoryAllocator::IMemoryBlockAccumulator {
    CMemoryAllocator::SMemoryStatistics m_stMemoryStatistics;
    size_t m_uNumberOfAllocatedMemoryBlocks;
    struct SMemoryInfo {
        char const* m_pszType;
        char const* m_pszFilename;
        uint16_t m_uLineNumber;
        bool m_bIsTemporarilyUntracked;
        size_t m_uSize;
        void* m_pvoidAddress;
    };
    SMemoryInfo m_astMemoryBlockInfo[g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS];
};
```

Description

This structure is used to report some information when the framework's Finalization occurs.

Constructors

Constructor	Description
• SFrameworkFinalizeInfo (see page 793)	Default Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~SFrameworkFinalizeInfo (see page 793)	Virtual destructor to avoid the warning caused by having another virtual function.

Legend

	Method
	virtual

Methods

CMemoryAllocator::IMemoryBlockAccumulator Class

CMemoryAllocator::IMemoryBlockAccumulator Class	Description
• A Accumulate (see page 488)	Notification function which receives the memory blocks being enumerated.

Legend

	Method
	abstract

2.16.2.1.1 - Data Members

2.16.2.1.1.1 - CFrameworkInitializer::SFrameworkFinalizeInfo::m_astMemoryBlockInfo Data Member

```
SMemoryInfo m_astMemoryBlockInfo[g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS];
```

Description

Array of up to MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS (see page 293) of currently allocated memory blocks. If there are less than MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS (see page 293) allocated blocks, the m_uNumberOfAllocatedMemoryBlocks (see page 793) describes the number of blocks in the array.

2.16.2.1.1.2 - CFrameworkInitializer::SFrameworkFinalizeInfo::m_stMemoryStatistics Data Member

A structure holding memory allocation statistics.

C++

```
CMemoryAllocator::SMemoryStatistics m_stMemoryStatistics;
```

2.16.2.1.1.3 -

CFrameworkInitializer::SFrameworkFinalizeInfo::m_uNumberOfAllocatedMemoryBlocks Data Member

```
size_t m_uNumberOfAllocatedMemoryBlocks;
```

Description

Count of total allocated memory blocks.

2.16.2.1.2 - Constructors

2.16.2.1.2.1 - CFrameworkInitializer::SFrameworkFinalizeInfo::SFrameworkFinalizeInfo Constructor

Default Constructor.

C++

```
SFrameworkFinalizeInfo();
```

Description

Default constructor for the SFrameworkFinalizeInfo that simply initializes structure values to their proper initial values.

2.16.2.1.3 - Destructors

2.16.2.1.3.1 - CFrameworkInitializer::SFrameworkFinalizeInfo::~SFrameworkFinalizeInfo Destructor

Virtual destructor to avoid the warning caused by having another virtual function.

C++

```
virtual ~SFrameworkFinalizeInfo();
```

2.16.2.2 - CFrameworkInitializer::SFrameworkFinalizeInfo::SMemoryInfo Struct

Holds a snapshot of memory at finalization time.

C++

```
struct SMemoryInfo {
    char const* m_pszType;
    char const* m_pszFilename;
    uint16_t m_uLineNumber;
    bool m_bIsTemporarilyUntracked;
    size_t m_uSize;
    void* m_pvoidAddress;
};
```

Description

This structure is used to hold a snapshot of the information contained in a CMemoryAllocator::CMemoryBlock (see page 485).

Members

Members	Description
char const* m_pszType;	A string representing the Type of data contained in the memory block.
char const* m_pszFilename;	The filename of the file in which the memory allocation was done.

<code>uint16_t m_uLineNumber;</code>	The line number in the file at which the memory allocation was done.
<code>bool m_bIsTemporarilyUntracked;</code>	A boolean value representing whether the memory block is marked as temporarily untracked.
<code>size_t m_uSize;</code>	The size of the memory block.
<code>void* m_pvoidAddress;</code>	A pointer to the memory block.

2.16.3 - Variables

This section documents the variables of the Sources/Startup folder.

2.16.3.1 - `g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS` Variable

```
const size_t g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS = 64;
```

Description

Defaults to 64 if undefined.

2.17 - Tls

This section documents the Sources/Tls folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 794)
- Enumerations (see page 815)

2.17.1 - Classes

This section documents the classes of the Sources/Tls folder.

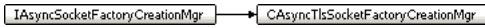
Classes

Class	Description
<code>CAsyncTlsSocketFactoryCreationMgr</code> (see page 794)	This class implements a TLS (see page 794) Socket Factory creation manager.
<code>CTls</code> (see page 796)	This class performs initialization of the TLS (see page 794) engine.
<code>CTlsContext</code> (see page 797)	The CTlsContext is the class that holds and manages various TLS (see page 794) handshaking parameters.
<code>CTlsSession</code> (see page 803)	Class that manages the TLS (see page 794) session identifier.
<code>IAsyncTlsOpenSsl</code> (see page 806)	Interface defining the basic methods accessible on asynchronous TLS (see page 794) sockets using the openSsl implementation.
<code>IAsyncTlsRenegotiation</code> (see page 807)	Interface defining the methods that perform and control TLS (see page 794) credentials renegotiation on a TLS (see page 794) socket.
<code>IAsyncTlsRenegotiationMgr</code> (see page 810)	Interface that receives event notifications from the <code>IAsyncTlsRenegotiation</code> (see page 807) interface.
<code>IAsyncTlsServerSocket</code> (see page 811)	Interface defining the basic methods accessible on asynchronous TLS (see page 794) server sockets.
<code>IAsyncTlsSocket</code> (see page 812)	Interface defining the basic methods accessible on asynchronous TLS (see page 794) sockets.
<code>IAsyncTlsSocketMgr</code> (see page 814)	This is the interface through which the asynchronous TLS (see page 794) sockets report their events.

2.17.1.1 - `CAsyncTlsSocketFactoryCreationMgr` Class

This class implements a TLS (see page 794) Socket Factory creation manager.

Class Hierarchy



C++

```
class CAsyncTlsSocketFactoryCreationMgr : public IAsyncSocketFactoryCreationMgr;
```

Description

This class implements a TLS (see page 794) Socket Factory creation manager. It implements the `IAsyncSocketFactoryCreationMgr` (see page 622) interface, and is automatically registered to the socket factory when `MXD_ECOM_ENABLE_SUPPORT` (see page 292), `MXD_SERVICING_THREAD_ENABLE_SUPPORT` (see page 306) and `MXD_TLS_OPENSSL` (see page 308) are enabled.

The `IAsyncSocketFactoryCreationMgr::EvCreationRequested` (see page 623) method creates a TLS (see page 794) asynchronous socket, and MUST only be called by the Socket Factory. The type of the asynchronous TLS (see page 794) socket to be created is provided as a sequence of network protocols starting from the highest level down to the lowest level. Each level is possibly followed by

arguments.

The only type of socket handled by this creation handler are TLS (see page 794) over TCP sockets. Moreover, the TLS (see page 794) token MUST have a "m=" option specifying whether a client or server socket is to be created. The TCP token can have an "m=" option in which case it must be the same as that of the TLS (see page 794) token. The "m=" option in the TCP token can be omitted.

If the type of the socket to be created is not handled by this creation manager, the method returns `resSI_FALSE`, so that the socket creation can be handled by another creation manager.

Examples of sockets handled by the TLS (see page 794) creation manager:

```
{
  "SIP", "TLS", "m=client", "TCP" }, 3
  {"SIP", "TLS", "m=client", "TCP", "m=client" }, 3
  {"SIP", "TLS", "m=server", "TCP" }, 3
  {"SIP", "TLS", "m=server", "TCP", "m=server" }, 3
```

Location

`Tls/CAsyncTlsSocketFactoryCreationMgr.h`

See Also

`IAsyncSocketFactoryCreationMgr` (see page 622), `CAsyncTlsSocket`, `CAsyncTlsServerSocket`

Constructors

Constructor	Description
◆ CAsyncTlsSocketFactoryCreationMgr (see page 795)	Default Constructor.

Legend

◆	Method
---	--------

Destructors

Destructor	Description
◆ ~CAsyncTlsSocketFactoryCreationMgr (see page 796)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
◆ ~CAsyncTlsSocketFactoryCreationMgr (see page 796)	

IAsyncSocketFactoryCreationMgr Class

IAsyncSocketFactoryCreationMgr Class	Description
◆ ~CAsyncTlsSocketFactoryCreationMgr (see page 623)	Notifies the manager that a new asynchronous socket must be created.

Legend

◆	Method
▼	virtual
▲	abstract

2.17.1.1.1 - Constructors

2.17.1.1.1 - CAsyncTlsSocketFactoryCreationMgr::CAsyncTlsSocketFactoryCreationMgr Constructor

Default Constructor.

C++

```
CAsyncTlsSocketFactoryCreationMgr();
```

2.17.1.1.2 - Destructors

2.17.1.1.2.1 - CAsyncTlsSocketFactoryCreationMgr::~CAsyncTlsSocketFactoryCreationMgr Destructor

Destructor.

C++

```
virtual ~CAsyncTlsSocketFactoryCreationMgr();
```

2.17.1.1.3 - Methods

2.17.1.1.3.1 - CAsyncTlsSocketFactoryCreationMgr::EvCreationRequested Method

```
virtual mxT_result EvCreationRequested(IN IECOMUnknown* pServicingThread, IN const char* const* apszType, IN
unsigned int uTypeSize, OUT IAsyncSocket** ppAsyncSocket);
```

2.17.1.2 - CTls Class

This class performs initialization of the TLS (see page 794) engine.

Class Hierarchy



C++

```
class CTls : public MXD_TLS_CTLs_CLASSNAME;
```

Description

CTls is a class used only by CAsyncTlsSocket and CAsyncTlsServerSocket. This is why all methods are private. No user shall ever have to directly use this class.

Location

Tls/CTls.h

See Also

CAsyncTlsSocket, CAsyncTlsServerSocket

Methods

Method	Description
Initialize (see page 796)	Must be called by CAsyncTlsSocket and CAsyncTlsServerSocket to initialize the TLS (see page 794) stack.
Instance (see page 796)	Returns the unique instance of CTls.
Uninitialize (see page 797)	Must be called by CAsyncTlsSocket and CAsyncTlsServerSocket to uninitialized the TLS (see page 794) stack.

Legend



2.17.1.2.1 - Methods

2.17.1.2.1.1 - CTls::Initialize Method

Must be called by CAsyncTlsSocket and CAsyncTlsServerSocket to initialize the TLS (see page 794) stack.

C++

```
void Initialize();
```

2.17.1.2.1.2 - CTls::Instance Method

Returns the unique instance of CTls (see page 796).

C++

```
static CTls* Instance();
```

Description

Returns the unique instance of the CTls (see page 796) class.

2.17.1.2.1.3 - CTls::Uninitialize Method

Must be called by CAsyncTlsSocket and CAsyncTlsServerSocket to uninitialized the TLS (see page 794) stack.

C++

```
void Uninitialize();
```

2.17.1.3 - CTlsContext Class

The CTlsContext is the class that holds and manages various TLS (see page 794) handshaking parameters.

Class Hierarchy

```
CTlsContext
```

C++

```
class CTlsContext;
```

Description

The CTlsContext is the class that holds and manages various TLS (see page 794) handshaking parameters.

The following parameters can be configured:

- **Certificate Chain:** The ordered list of certificates that is sent to the peer. See CCertificateChain (see page 691) for more information.
- **Cipher:** The Cipher is a string that specifies which cipher algorithm suites must be used for the connection. The default is "ALL", which specifies that all supported cipher suites are allowed.
- **Peer Authentication:** Specifies whether or not the peer must be authenticated. If true, a certificate request is sent to the peer. The received certificate is then evaluated for trust. The default is peer authenticated.
- **Protocol Versions:** Specifies which protocol versions are allowed. This could be SSL 3.0 or TLS (see page 794) 1.0.
- **Trusted Certificates:** Specifies a list of root certificates that are trusted. If the peer presents a certificate that does inherit from one of them, the connection handshaking is accepted.

Location

Tls/CTlsContext.h

See Also

CCertificate (see page 677), CCertificateChain (see page 691)

Constructors

Constructor	Description
~CTlsContext (see page 798)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
~CTlsContext (see page 799)	Destructor.

Legend

	Method
	virtual

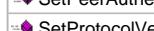
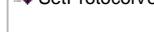
Operators

Operator	Description
 = (see page 803)	Assignment operator.

Legend

	Method
--	--------

Methods

Method	Description
 GetCertificateChain (see page 799)	Returns the certificates chain configured for this context.
 GetCiphers (see page 799)	Returns the allowed ciphers.
 GetEphemeralDiffieHellman (see page 800)	Returns the current Diffie-Hellman configuration.
 GetPeerAuthentication (see page 800)	Returns true if the peer must be authenticated, false otherwise.
 GetProtocolVersions (see page 800)	Returns the protocol versions configured within the context.
 GetTrustedCertificates (see page 800)	Returns the trusted certificates configured within the context.
 SetCertificateChain (see page 800)	Sets the current certificates chain.
 SetCiphers (see page 801)	Configures the allowed ciphers that may be used in the capability exchange. Defaults to "ALL".
 SetEphemeralDiffieHellman (see page 802)	Sets the current Diffie-Hellman configuration.
 SetPeerAuthentication (see page 802)	Configures if the peer must be authenticated. Defaults to true.
 SetProtocolVersions (see page 802)	Sets the current protocol versions. Defaults to eTLS_VERSION_SSL_3_0 and eTLS_VERSION_TLS_1_0.
 SetTrustedCertificates (see page 802)	Sets the current trusted certificates.

Legend

	Method
--	--------

2.17.1.3.1 - Constructors**2.17.1.3.1.1 - CTlsContext**

Constructor.

C++

```
CTlsContext();
```

Description

Constructor.

2.17.1.3.1.1.2 - CTlsContext::CTlsContext Constructor

Copy constructor. Reference version.

C++

```
CTlsContext(IN const CTlsContext& rTlsContext);
```

Parameters

Parameters	Description
IN const CTlsContext& rTlsContext	Reference to the TLS (see page 794) context to copy.

Description

Copy constructor.

2.17.1.3.1.1.3 - CTlsContext::CTlsContext Constructor

Copy constructor. Pointer version.

C++

```
CTlsContext( IN const CTlsContext* pTlsContext );
```

Parameters

Parameters	Description
IN const CTlsContext* pTlsContext	Pointer to the TLS (see page 794) context to copy.

Description

Copy constructor.

2.17.1.3.2 - Destructors

2.17.1.3.2.1 - CTlsContext::~CTlsContext Destructor

Destructor.

C++

```
virtual ~CTlsContext();
```

Description

Destructor.

2.17.1.3.3 - Methods

2.17.1.3.3.1 - CTlsContext::GetCertificateChain Method

Returns the certificates chain configured for this context.

C++

```
void GetCertificateChain(OUT CCertificateChain* pCertificateChain) const;
```

Parameters

Parameters	Description
OUT CCertificateChain* pCertificateChain	Pointer to contain the certificate chain.

Description

Gets the certificate chain stored in the TLS (see page 794) context.

2.17.1.3.3.2 - CTlsContext::GetCiphers Method

Returns the allowed ciphers.

C++

```
void GetCiphers(OUT CString* pstrCiphers) const;
```

Parameters

Parameters	Description
OUT CString* pstrCiphers	Pointer to contain the cipher string.

Description

Gets the allowed ciphers.

See Also

SetCiphers (see page 801)

2.17.1.3.3.3 - CTlsContext::GetEphemeralDiffieHellman Method

Returns the current Diffie-Hellman configuration.

C++

```
void GetEphemeralDiffieHellman(OUT CDiffieHellman* pEphemeralDiffieHellman) const;
```

Parameters

Parameters	Description
OUT CDiffieHellman* pEphemeralDiffieHellman	Pointer to contain the current CDiffieHellman (See page 340).

Description

Gets the current Diffie-Hellman configuration.

2.17.1.3.3.4 - CTlsContext::GetPeerAuthentication Method

Returns true if the peer must be authenticated, false otherwise.

C++

```
void GetPeerAuthentication(OUT bool* pbAuthenticatePeer) const;
```

Parameters

Parameters	Description
OUT bool* pbAuthenticatePeer	Pointer to contain the peer authentication flag.

Description

Returns true if the peer must be authenticated, false otherwise.

2.17.1.3.3.5 - CTlsContext::GetProtocolVersions Method

Returns the protocol versions configured within the context.

C++

```
void GetProtocolVersions(OUT CVector<ETlsVersion>* pvecProtocolVersions) const;
```

Parameters

Parameters	Description
OUT CVector<ETlsVersion>* pvecProtocolVersions	Pointer to contain the protocol versions.

Description

Returns the protocol versions configured within the context.

2.17.1.3.3.6 - CTlsContext::GetTrustedCertificates Method

Returns the trusted certificates configured within the context.

C++

```
void GetTrustedCertificates(OUT CVector<CCertificate>* pvecTrustedCertificates) const;
```

Parameters

Parameters	Description
OUT CVector<CCertificate>* pvecTrustedCertificates	Pointer to contain the trusted certificates.

Description

Returns the trusted certificates configured within the context.

2.17.1.3.3.7 - CTlsContext::SetCertificateChain Method

Sets the current certificates chain.

C++

```
void SetCertificateChain(IN CCertificateChain* pCertificateChain);
```

Parameters

Parameters	Description
IN CCertificateChain* pCertificateChain	Pointer containing the certificate chain.

Description

Sets the certificate chain to store in the TLS (see page 794) context.

2.17.1.3.3.8 - CTlsContext::SetCiphers Method

Configures the allowed ciphers that may be used in the capability exchange. Defaults to "ALL"

C++

```
mxt_result SetCiphers(IN const CString& rstrCiphers);
```

Parameters

Parameters	Description
IN const CString& rstrCiphers	A string representing a set of rules that must be interpreted to discover which ciphers are allowed, and which are not.

Description

This method is used to specify in a generic way the cipher suites to use.

The cipher list consists of one or more cipher strings separated by colons. Unknown cipher strings are silently discarded. Cipher strings can take several forms.

- It can consist of a single cipher suite such as RC4-SHA.
- It can represent a list of cipher suites containing a certain algorithm, or cipher suites of a certain type. For example, SHA-1 represents all cipher suites using the digest algorithm SHA-1.

Lists of cipher suites can be combined into a single cipher string by using the + character. This is used as a logical AND operation. For example, SHA1+DES represents all cipher suites containing the SHA-1 and the DES algorithms.

Each cipher string can be optionally preceded by the characters !, -, or +.

If ! is used, then the ciphers are permanently deleted from the list. The ciphers detected can never reappear in the list even if they are explicitly stated.

If - is used, then the ciphers are deleted from the list, but some or all of the ciphers can be added again by later options.

If + is used, then the ciphers are moved to the end of the list. This option does not add any new cipher, it just moves matching existing ones.

If none of these characters are present, then the string is just interpreted as a list of ciphers to be appended to the current preference list. If the list includes ciphers already present, they are ignored; they will not be moved to the end of the list.

The following is a list of all permitted cipher strings and their meanings.

- ALL: All cipher suites except those that contain the eNULL cipher, which must be explicitly enabled.
- DEFAULT: The default cipher list. Must be the first cipher string specified.
- EXPORT: Cipher suites that contain a symmetric cipher with key lengths equal to 40 or 56 bits mark as exportable.
- HIGH: Cipher suites that contain a symmetric cipher with key lengths larger than 128 bits.
- LOW: Cipher suites that contain a symmetric cipher with key lengths equal to 64 or 56 bits and excluding exportable ciphers.
- MEDIUM: Cipher suites that contain a symmetric cipher with key lengths equal to 128 bits.
- SHA1: Cipher suites that contain SHA-1 as digest algorithm.
- MD5: Cipher suites which contain MD5 as digest algorithm.
- aDH: Cipher suites that offer DH authentication. The certificates carry DH keys.
- aDSS: Cipher suites that offer DSS authentication. The certificates carry DSA keys.

- aNULL: Cipher suites that offer no authentication.
- aRSA: Cipher suites that offer RSA authentication. The certificates carry RSA keys.
- eNULL: Cipher suites that offer no encryption.
- kDHd: Cipher suites that offer DH key agreement and DH certificate signed by CA with DSS.
- kDHR: Cipher suites that offer DH key agreement and DH certificate signed by CA with RSA.
- kEDH: Cipher suites that offer ephemeral DH key agreement.
- kRSA: Cipher suites that offer RSA key exchange.

Example

```
"ALL:laDH:!LOW:!MD5"
```

2.17.1.3.3.9 - CTlsContext::SetEphemeralDiffieHellman Method

Sets the current Diffie-Hellman configuration.

C++

```
void SetEphemeralDiffieHellman(IN const CDiffieHellman* pEphemeralDiffieHellman);
```

Parameters

Parameters	Description
IN const CDiffieHellman* pEphemeralDiffieHellman	Pointer containing the current CDiffieHellman (See page 340).

Description

Sets the current Diffie-Hellman configuration.

2.17.1.3.3.10 - CTlsContext::SetPeerAuthentication Method

Configures if the peer must be authenticated. Defaults to true.

C++

```
void SetPeerAuthentication(IN bool bAuthenticatePeer);
```

Parameters

Parameters	Description
pbAuthenticatePeer	Pointer containing the peer authentication flag.

Description

Enables/disables peer authentication.

2.17.1.3.3.11 - CTlsContext::SetProtocolVersions Method

Sets the current protocol versions. Defaults to eTLS_VERSION_SSL_3_0 and eTLS_VERSION_TLS_1_0.

C++

```
void SetProtocolVersions(IN const CVector<ETlsVersion>* pvecProtocolVersions);
```

Parameters

Parameters	Description
IN const CVector<ETlsVersion>* pvecProtocolVersions	Pointer containing the protocol versions.

Description

Sets the current protocol versions. Defaults to eTLS_VERSION_SSL_3_0 and eTLS_VERSION_TLS_1_0.

2.17.1.3.3.12 - CTlsContext::SetTrustedCertificates Method

Sets the current trusted certificates.

C++

```
void SetTrustedCertificates(IN const CVector<CCertificate>* pvecTrustedCertificates);
```

Parameters

Parameters	Description
IN const CVector<CCertificate>* pvecTrustedCertificates	Pointer containing the trusted certificates.

Description

Sets the current trusted certificates. This removes any previously set certificate.

2.17.1.3.4 - Operators

2.17.1.3.4.1 - CTlsContext::= Operator

Assignment operator.

C++

```
CTlsContext& operator =(IN const CTlsContext& rTlsContext);
```

Parameters

Parameters	Description
IN const CTlsContext& rTlsContext	Pointer to the TLS (see page 794) context to copy.

Returns

A reference to the assigned CTlsContext (see page 797).

Description

Assigns the right hand CTlsContext (see page 797) to the left hand one.

2.17.1.4 - CTlsSession Class

Class that manages the TLS (see page 794) session identifier.

Class Hierarchy



C++

```
class CTlsSession : public MXD_TLS_CTLSESSION_CLASSNAME;
```

Description

CTlsSession is the class that manages the TLS (see page 794) session identifier. It serves as a container for the session identifier. The session identifier is used to re-establish a connection without going through the whole complete handshaking.

Location

Tls/CTlsSession.h

Constructors

Constructor	Description
~CTlsSession (see page 804)	Constructor.

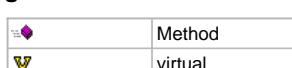
Legend



Destructors

Destructor	Description
~CTlsSession (see page 805)	Destructor.

Legend



Operators

Operator	Description
<code>=</code> (see page 806)	Assignment operator.

Legend

	Method
--	--------

Methods

Method	Description
<code>GetId</code> (see page 805)	Returns the session's ID.
<code>Restore</code> (see page 805)	Restores a TLS (see page 794) session from binary data.
<code>Store</code> (see page 805)	Stores a TLS (see page 794) session in binary data.

Legend

	Method
--	--------

2.17.1.4.1 - Constructors

2.17.1.4.1.1 - CTlsSession

2.17.1.4.1.1.1 - CTlsSession::CTlsSession Constructor

Constructor.

C++

```
CTlsSession();
```

Description

Constructor

2.17.1.4.1.1.2 - CTlsSession::CTlsSession Constructor

Copy constructor. Reference version.

C++

```
CTlsSession(IN const CTlsSession& rTlsSession);
```

Parameters

Parameters	Description
<code>IN const CTlsSession& rTlsSession</code>	Reference to the session to copy.

Description

Copy constructor

2.17.1.4.1.1.3 - CTlsSession::CTlsSession Constructor

Copy constructor. Pointer version.

C++

```
CTlsSession(IN const CTlsSession* pTlsSession);
```

Parameters

Parameters	Description
<code>IN const CTlsSession* pTlsSession</code>	Pointer to the session to copy.

Description

Copy constructor

2.17.1.4.2 - Destructors

2.17.1.4.2.1 - CTlsSession::~CTlsSession Destructor

Destructor.

C++

```
virtual ~CTlsSession();
```

Description

Destructor.

2.17.1.4.3 - Methods

2.17.1.4.3.1 - CTlsSession::GetId Method

Returns the session's ID.

C++

```
mxt_result GetId(OUT CBlob* pblobId) const;
```

Parameters

Parameters	Description
OUT CBlob* pblobId	The session ID.

Returns

resS_OK: pblobId holds a valid session ID.

resFE_INVALID_ARGUMENT: pblobId is NULL.

resFE_INVALID_STATE: No SSL session has been set.

Description

Returns the session's ID.

2.17.1.4.3.2 - CTlsSession::Restore Method

Restores a TLS (see page 794) session from binary data.

C++

```
mxt_result Restore(IN CBlob* pBlob);
```

Parameters

Parameters	Description
IN CBlob* pBlob	Pointer to a blob containing the session information.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Restores the TLS (see page 794) session information from a blob.

2.17.1.4.3.3 - CTlsSession::Store Method

Stores a TLS (see page 794) session in binary data.

C++

```
mxt_result Store(OUT CBlob* pBlob) const;
```

Parameters

Parameters	Description
OUT CBlob* pBlob	Pointer to a blob to contain the session information.

Returns

- resS_OK
- resFE_FAIL
- resFE_INVALID_ARGUMENT

Description

Stores the TLS (see page 794) session information to a blob.

2.17.1.4.4 - Operators

2.17.1.4.4.1 - CTlsSession::= Operator

Assignment operator.

C++

```
CTlsSession& operator =(IN const CTlsSession& rTlsSession);
```

Parameters

Parameters	Description
IN const CTlsSession& rTlsSession	Reference to the TLS (see page 794) session to assign.

Returns

A reference to the assigned CTlsSession (see page 803).

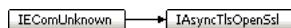
Description

Assigns the right hand session to the left hand one.

2.17.1.5 - IAsyncTlsOpenSsl Class

Interface defining the basic methods accessible on asynchronous TLS (see page 794) sockets using the openSsl implementation.

Class Hierarchy



C++

```
class IAsyncTlsOpenSsl : public IEComUnknown;
```

Description

Interface defining the most basic methods accessible on asynchronous TLS (see page 794) sockets using the openSsl implementation.

Location

Tls/IAsyncTlsOpenSsl.h

Methods

Method	Description
◆ A GetSsl (see page 807)	Gets the OpenSSL SSL object for this socket.

IECOMUnknown Class

IECOMUnknown Class	Description
◆ A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
◆ A QueryIf (see page 418)	Queries an object for a supported interface.
◆ A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.17.1.5.1 - Methods**2.17.1.5.1.1 - IAsyncTlsOpenSsl::GetSsl Method**

Gets the OpenSSL SSL object for this socket.

C++

```
virtual mxt_result GetSsl(OUT const SSL** ppSsl) const = 0;
```

Parameters

Parameters	Description
OUT const SSL** ppSsl	The OpenSSL SSL object's pointer when resS_OK is returned.

Returns

- resS_OK: The SSL object has successfully been retrieved.
- resFE_INVALID_ARGUMENT: ppSsl is NULL.
- resFE_INVALID_STATE: The OpenSSL SSL object is not available. This occurs before the TLS (see page 794) connection is established.

Description

Gets the OpenSSL SSL object for this socket.

2.17.1.6 - IAsyncTlsRenegotiation Class

Interface defining the methods that perform and control TLS (see page 794) credentials renegotiation on a TLS (see page 794) socket.

Class Hierarchy**C++**

```
class IAsyncTlsRenegotiation : public IEComUnknown;
```

Description

Interface defining the methods that perform and control TLS (see page 794) credentials renegotiation on a TLS (see page 794) socket. This means performing manual renegotiation, setting the parameters needed to perform automatic renegotiation, and setting the manager for the interface.

Events related to this interface are reported through the IAsyncTlsRenegotiationMgr (see page 810) interface.

Location

Tls/IAsyncTlsRenegotiation.h

See Also

IAsyncTlsRenegotiationMgr (see page 810), CAAsyncTlsSocket

Methods

Method	Description
  EnableAllRenegotiationNotifications (see page 808)	
  RenegotiateA (see page 808)	Manually starts the renegotiation of the security parameters.
  SetAsyncTlsRenegotiationMgr (see page 808)	Sets the TLS (see page 794) socket's renegotiation manager.
  SetAutoRenegotiationThresholdInByte (see page 809)	Sets the maximum number of bytes allowed between renegotiations.
  SetAutoRenegotiationThresholdInTimeMs (see page 809)	Sets the maximum timeout allowed between renegotiations.
  SetAutoRenegotiationTimeoutMs (see page 809)	Sets the maximum timeout to wait for renegotiation to succeed.

IEComUnknown Class

IEComUnknown Class	Description
•   AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
•   QueryIf (see page 418)	Queries an object for a supported interface.
•   ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.17.1.6.1 - Methods**2.17.1.6.1.1 - IAsyncTlsRenegotiation::EnableAllRenegotiationNotifications Method**

```
virtual mxt_result EnableAllRenegotiationNotifications(IN bool bEnable) = 0;
```

Returns

- resS_OK: Method processed successfully.

Description

Controls events notifications through the IAsyncTlsRenegotiationMgr::EvAsyncTlsRenegotiationMgrRenegotiated (see page 810) manager when renegotiation succeeds. When disabled, only the manual renegotiation event (when RenegotiateA (see page 808) is used) is reported. Not calling this method has the same behavior as using it with a false argument.

See Also

RenegotiateA (see page 808)

2.17.1.6.1.2 - IAsyncTlsRenegotiation::RenegotiateA Method

Manually starts the renegotiation of the security parameters.

C++

```
virtual mxt_result RenegotiateA(IN uint64_t uTimeoutMs) = 0;
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs	The maximum time period allowed for renegotiation. Once the timeout has elapsed, an error is triggered.

Returns

- resS_OK: Method processed successfully.

Description

Manually starts the renegotiation of the security parameters.

See Also

EnableAllRenegotiationNotifications (see page 808)

2.17.1.6.1.3 - IAsyncTlsRenegotiation::SetAsyncTlsRenegotiationMgr Method

Sets the TLS (see page 794) socket's renegotiation manager.

C++

```
virtual mxt_result SetAsyncTlsRenegotiationMgr(IN IAsyncTlsRenegotiationMgr* pAsyncTlsRenegotiationMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncTlsRenegotiationMgr* pAsyncTlsRenegotiationMgr	Socket manager to set. It cannot be NULL.

Returns

- resFE_INVALID_ARGUMENT: pAsyncTlsSocketMgr is NULL.
- resFE_INVALID_STATE: The TLS (see page 794) socket's renegotiation manager is already set.
- resS_OK: Method processed successfully.

Description

Sets the TLS (see page 794) socket's renegotiation manager.

2.17.1.6.1.4 - IAsyncTlsRenegotiation::SetAutoRenegotiationThresholdInByte Method

Sets the maximum number of bytes allowed between renegotiations.

C++

```
virtual mxt_result SetAutoRenegotiationThresholdInByte(IN uint32_t uByte) = 0;
```

Parameters

Parameters	Description
IN uint32_t uByte	The maximum number of bytes that may be exchanged in one direction before automatic renegotiation starts ('0' disables automatic renegotiation).

Returns

- resS_OK: Method processed successfully.

Description

The user can specify the maximum number of bytes that may be received in a direction before automatic renegotiation is triggered. Set it to 0 to disable it.

See Also

EnableAllRenegotiationNotifications (see page 808)

2.17.1.6.1.5 - IAsyncTlsRenegotiation::SetAutoRenegotiationThresholdInTimeMs Method

Sets the maximum timeout allowed between renegotiations.

C++

```
virtual mxt_result SetAutoRenegotiationThresholdInTimeMs(IN uint64_t uTimeMs) = 0;
```

Parameters

Parameters	Description
IN uint64_t uTimeMs	The maximum time allowed between renegotiations ('0' disables automatic renegotiation).

Returns

- resS_OK: Method processed successfully.

Description

The user can specify the interval after which the automatic renegotiation is triggered. Set it to 0 to disable it.

See Also

EnableAllRenegotiationNotifications (see page 808)

2.17.1.6.1.6 - IAsyncTlsRenegotiation::SetAutoRenegotiationTimeoutMs Method

Sets the maximum timeout to wait for renegotiation to succeed.

C++

```
virtual mxt_result SetAutoRenegotiationTimeoutMs(IN uint64_t uTimeoutMs) = 0;
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs	The maximum number of ms to wait until the renegotiation fails ('0' disables the timeout).

Returns

- resS_OK: Method processed successfully.

Description

The user can specify the maximum time period allowed for automatic renegotiation to succeed. Once the timeout has elapsed, an error is triggered. Set it to 0 to disable it.

2.17.1.7 - IAsyncTlsRenegotiationMgr Class

Interface that receives event notifications from the IAsyncTlsRenegotiation (see page 807) interface.

Class Hierarchy

IAsyncTlsRenegotiationMgr

C++

```
class IAsyncTlsRenegotiationMgr;
```

Description

Interface that receives event notifications from the IAsyncTlsRenegotiation (see page 807) interface. All events are reported asynchronously with respect to the manager's execution context.

Location

Tls/IAsyncTlsRenegotiationMgr.h

See Also

IAsyncTlsRenegotiation (see page 807), CAsyncTlsSocket

Methods

Method	Description
◆ EvAsyncTlsRenegotiationMgrRenegotiated (see page 810)	Notifies the successful renegotiation.

Legend

◆	Method
▲	abstract

2.17.1.7.1 - Methods

2.17.1.7.1.1 - IAsyncTlsRenegotiationMgr::EvAsyncTlsRenegotiationMgrRenegotiated Method

Notifies the successful renegotiation.

C++

```
virtual void EvAsyncTlsRenegotiationMgrRenegotiated(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket.

Description

This event is generated upon the successful completion of the renegotiation.

Depending on the setting of IAsyncTlsRenegotiation::EnableAllRenegotiationNotifications (see page 808), it may be reported only for manual renegotiation (explicit call to IAsyncTlsRenegotiation::RenegotiateA (see page 808)) or for all renegotiations.

Unsuccessful renegotiation attempts are reported through the IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected (see page 625) event.

See Also

[IAsyncTlsRenegotiation::RenegotiateA](#) (see page 808), [IAsyncTlsRenegotiation::EnableAllRenegotiationNotifications](#) (see page 808), [IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected](#) (see page 625)

2.17.1.8 - IAsyncTlsServerSocket Class

Interface defining the basic methods accessible on asynchronous TLS (see page 794) server sockets.

Class Hierarchy**C++**

```
class IAsyncTlsServerSocket : public IEComUnknown;
```

Description

Interface defining the most basic methods accessible on asynchronous TLS (see page 794) server sockets. This means the retrieving and setting of the TLS (see page 794) Context that will be given to the accepted sockets.

Location

Tls/IAsyncTlsServerSocket.h

See Also

[CAsyncTlsServerSocket](#)

Methods

Method	Description
• A GetAcceptedTlsContext (see page 811)	Gets a copy of the CTlsContext (see page 797) that will be used for accepted TLS (see page 794) sockets.
• A SetAcceptedTlsContext (see page 812)	Sets the current TLS (see page 794) context that will be used on accepted TLS (see page 794) sockets.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

• A	Method
A	abstract

2.17.1.8.1 - Methods**2.17.1.8.1.1 - IAsyncTlsServerSocket::GetAcceptedTlsContext Method**

Gets a copy of the CTlsContext (see page 797) that will be used for accepted TLS (see page 794) sockets.

C++

```
virtual mxt_result GetAcceptedTlsContext(OUT CTlsContext* pAcceptedTlsContext) const = 0;
```

Parameters

Parameters	Description
OUT CTlsContext* pAcceptedTlsContext	Pointer to a CTlsContext (see page 797) object that will receive the copy. It cannot be NULL.

Returns

resS_OK: The CTlsContext (see page 797) has been successfully retrieved. resFE_INVALID_ARGUMENT: pAcceptedTlsContext is NULL.

Description

Gets a copy of the CTlsContext (see page 797) that will be used for accepted TLS (see page 794) sockets.

2.17.1.8.1.2 - IAsyncTlsServerSocket::SetAcceptedTlsContext Method

Sets the current TLS (see page 794) context that will be used on accepted TLS (see page 794) sockets.

C++

```
virtual mxt_result SetAcceptedTlsContext(IN const CTlsContext* pTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CTlsContext* pTlsContext	The new TLS (see page 794) context to use. It cannot be NULL.

Returns

- resFE_INVALID_ARGUMENT: pTlsContext is NULL.
- resS_OK: Method processed successfully.

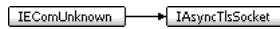
Description

Sets the current TLS (see page 794) context that will be used on accepted TLS (see page 794) sockets.

2.17.1.9 - IAsyncTlsSocket Class

Interface defining the basic methods accessible on asynchronous TLS (see page 794) sockets.

Class Hierarchy



C++

```
class IAsyncTlsSocket : public IECOMUnknown;
```

Description

Interface defining the most basic methods accessible on asynchronous TLS (see page 794) sockets. This means retrieving and setting the TLS (see page 794) Context, setting the TLS (see page 794) Session, setting the TLS (see page 794) asynchronous socket's manager, and completing the peer credentials approval.

Events related to this interface are reported through the IAsyncTlsSocketMgr (see page 814) interface.

Location

Tls/IAsyncTlsSocket.h

See Also

IAsyncTlsSocketMgr (see page 814), CAsyncTlsSocket

Methods

Method	Description
• A GetTlsContext (see page 813)	Gets a copy of the CTlsContext (see page 797) used by this context.
• A SetAsyncTlsSocketMgr (see page 813)	Sets the TLS (see page 794) socket's manager.
• A SetTlsContext (see page 813)	Sets the current TLS (see page 794) context.
• A SetTlsSession (see page 814)	Sets the TLS (see page 794) session used for TLS (see page 794) handshaking.
• A TlsHandshakingApprovalCompletedA (see page 814)	Notifies that the handshaking approval is complete.

IEComUnknown Class

IEComUnknown Class	Description
• A AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
• A QueryIf (see page 418)	Queries an object for a supported interface.
• A ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.17.1.9.1 - Methods**2.17.1.9.1.1 - IAsyncTlsSocket::GetTlsContext Method**

Gets a copy of the CTlsContext (see page 797) used by this context.

C++

```
virtual mxt_result GetTlsContext(OUT CTlsContext* pTlsContext) const = 0;
```

Parameters

Parameters	Description
OUT CTlsContext* pTlsContext	Pointer to a CTlsContext (see page 797) object that will receive the copy.

Returns

resS_OK: The CTlsContext (see page 797) has been successfully retrieved. resFE_INVALID_ARGUMENT: pTlsContext is NULL.

Description

Gets a copy of the CTlsContext (see page 797) used by this socket.

2.17.1.9.1.2 - IAsyncTlsSocket::SetAsyncTlsSocketMgr Method

Sets the TLS (see page 794) socket's manager.

C++

```
virtual mxt_result SetAsyncTlsSocketMgr(IN IAsyncTlsSocketMgr* pAsyncTlsSocketMgr) = 0;
```

Parameters

Parameters	Description
IN IAsyncTlsSocketMgr* pAsyncTlsSocketMgr	Socket manager to set. It cannot be NULL.

Returns

- resFE_INVALID_ARGUMENT: pAsyncTlsSocketMgr is NULL.
- resFE_INVALID_STATE: The TLS (see page 794) socket's manager is already set.
- resS_OK: Method processed successfully.

Description

Sets the TLS (see page 794) socket's manager.

2.17.1.9.1.3 - IAsyncTlsSocket::SetTlsContext Method

Sets the current TLS (see page 794) context.

C++

```
virtual mxt_result SetTlsContext(IN const CTlsContext* pTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CTlsContext* pTlsContext	The new TLS (see page 794) context to use. It cannot be NULL.

Returns

- resFE_INVALID_ARGUMENT: pTlsContext is NULL.
- resFE_INVALID_STATE: A renegotiation is in progress.
- resS_OK: Method processed successfully.

Description

This method sets the current TLS (see page 794) context. The TLS (see page 794) context is used for TLS (see page 794) handshaking. Thus, it is important to set it before any TLS (see page 794) action is performed. Failure to do so will result in an error being reported.

2.17.1.9.1.4 - IAsyncTlsSocket::SetTlsSession Method

Sets the TLS (see page 794) session used for TLS (see page 794) handshaking.

C++

```
virtual mxt_result SetTlsSession(IN const CTlsSession* pTlsSession) = 0;
```

Parameters

Parameters	Description
IN const CTlsSession* pTlsSession	The session used for TLS (see page 794) handshaking. It cannot be NULL.

Returns

- resFE_INVALID_ARGUMENT: pTlsSession is NULL.
- resS_OK: Method processed successfully.

Description

To initiate handshaking, the TLS (see page 794) protocol allows the client to send a TLS (see page 794) session identifier, which bypasses the key negotiation phase. This is a speed up trick. This call must be done prior to TLS (see page 794) handshaking to be effective.

2.17.1.9.1.5 - IAsyncTlsSocket::TlsHandshakingApprovalCompletedA Method

Notifies that the handshaking approval is complete.

C++

```
virtual mxt_result TlsHandshakingApprovalCompletedA(IN mxt_result resApproval) = 0;
```

Parameters

Parameters	Description
IN mxt_result resApproval	mxt_result (see page 92) containing the approval.

Returns

- resS_OK: Always.

Description

Notifies that the handshaking approval is complete. This means that the application has validated the peer credentials. These credentials are provided by the IAsyncTlsSocketMgr::EvAsyncTlsSocketMgrTlsHandshakingCompletedAwaitingApproval (see page 815) event. In most cases, this method will be called from that event handler.

2.17.1.10 - IAsyncTlsSocketMgr Class

This is the interface through which the asynchronous TLS (see page 794) sockets report their events.

Class Hierarchy

```
[IAsyncTlsSocketMgr]
```

C++

```
class IAsyncTlsSocketMgr;
```

Description

This is the interface through which the asynchronous TLS (see page 794) sockets report their events. All events are reported asynchronously with respect to the manager's execution context.

Location

Tls/IAsyncTlsSocketMgr.h

Methods

Method	Description
• A EvAsyncTlsSocketMgrNewTlsSession (see page 815)	Notifies of the chosen session parameters.
• A EvAsyncTlsSocketMgrTlsHandshakingCompletedAwaitingApproval (see page 815)	Notifies that the TLS (see page 794) handshaking is completed and gives the manager the opportunity to do some more validation prior to IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected (see page 602) or IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted (see page 611) notifications.

Legend

•	Method
A	abstract

2.17.1.10.1 - Methods

2.17.1.10.1.1 - IAsyncTlsSocketMgr::EvAsyncTlsSocketMgrNewTlsSession Method

Notifies of the chosen session parameters.

C++

```
virtual void EvAsyncTlsSocketMgrNewTlsSession(IN mxt_opaque opq, IN const CTlsSession* pTlsSession) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket.
IN const CTlsSession* pTlsSession	A pointer to a session description class. The user must never keep a reference to this object. It must create a copy of the underlying object if it wants to keep it.

Description

This is the event generated by an asynchronous TLS (see page 794) handshake completion. It is a means to publish current session descriptions. Only the entity acting as the TLS (see page 794) client side receives this event.

2.17.1.10.1.2 -

IAsyncTlsSocketMgr::EvAsyncTlsSocketMgrTlsHandshakingCompletedAwaitingApproval Method

Notifies that the TLS (see page 794) handshaking is completed and gives the manager the opportunity to do some more validation prior to IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected (see page 602) or IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted (see page 611) notifications.

C++

```
virtual void EvAsyncTlsSocketMgrTlsHandshakingCompletedAwaitingApproval(IN mxt_opaque opq, IN const CCertificateChain* pPeerCertificateChain, IN IAsyncTlsSocket* pAsyncTlsSocket) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	Opaque value associated with the socket.
IN const CCertificateChain* pPeerCertificateChain	The certificates just received from the peer. The manager must create a copy of the certificate chain if it wants to keep it.
IN IAsyncTlsSocket* pAsyncTlsSocket	Pointer to an IAsyncTlsSocket (see page 812) interface representing the socket.

Description

This is the event generated by an asynchronous TLS (see page 794) handshake completion. It is a mean for the manager to do more specific validations on the peer credentials.

See Also

IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected (see page 602),
IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted (see page 611)

2.17.2 - Enumerations

This section documents the enumerations of the Sources/Tls folder.

Enumerations

Enumeration	Description
ETlsVersion (see page 816)	SSL/TLS (see page 794) protocol versions.

2.17.2.1 - ETlsVersion Enumeration

SSL/TLS (see page 794) protocol versions.

C++

```
enum ETlsVersion {
    eTLS_VERSION_SSL_3_0,
    eTLS_VERSION_TLS_1_0
};
```

Description

SSL/TLS (see page 794) protocol versions supported by M5T Framework.

Members

Members	Description
eTLS_VERSION_SSL_3_0	SSL v3.0
eTLS_VERSION_TLS_1_0	TLS (see page 794) v1.0

2.18 - Time

This section documents the Sources/Time folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 816)
- Enumerations (see page 838)
- Variables (see page 839)

2.18.1 - Classes

This section documents the classes of the Sources/Time folder.

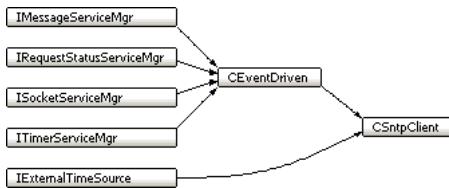
Classes

Class	Description
CSntpClient (see page 816)	Implements a SNTP client to connect to a SNTP or NTP server.
CTime (see page 822)	Class for retrieving the date and time.
CTimeZone (see page 832)	Implements the conversion between time zones and the handling of daylight saving time.
IExternalTimeSource (see page 837)	Base class to be inherited by every class that wants to be a time source.

2.18.1.1 - CSntpClient Class

Implements a SNTP client to connect to a SNTP or NTP server.

Class Hierarchy



C++

```
class CSntpClient : public CEventDriven, public IExternalTimeSource;
```

Description

CSntpClient implements a SNTP client that connects to a SNTP or NTP server and gets the new time. It does not keep the time, it only retrieves the time from the server. CSntpClient follows RFC 1769.

There are a number of settings that can statically change.

The global timeout, named only timeout, represents the maximum time that the client is allowed to wait for an answer when it executes a query.

The SntpPacketSentToServer is the number of packets that the client sends to the server. This might prove useful if there are a lot of packets lost on the network.

Constructors

Constructor	Description
• CSntpClient (see page 818)	Constructor.

CEventDriven Class

CEventDriven Class	Description
• CEventDriven (see page 764)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~CSntpClient (see page 818)	Destructor.

CEventDriven Class

CEventDriven Class	Description
• ~CEventDriven (see page 764)	Destructor.

Legend

•	Method
▼	virtual

Methods

Method	Description
• Activate (see page 818)	Activates the thread mechanism of the SNTP client.
• ClearServerList (see page 819)	Clears the SNTP servers list.
• ▼ GetTimeS (see page 819)	Gets the time.
• InsertServer (see page 819)	Inserts a new server into the servers list.
• QueryServer (see page 820)	Requests the current date and time from the registered servers.
• SetSntpPacketSentToServer (see page 822)	Sets the number of packets sent to a server at each request.
• SetTimeoutMs (see page 822)	Sets the default connection timeout.
• Terminate (see page 822)	Terminates the thread mechanism of the SNTP client.

IExternalTimeSource Class

IExternalTimeSource Class	Description
• ▲ GetTimeS (see page 838)	Gets the time.

CEventDriven Class

CEventDriven Class	Description
• Activate (see page 764)	Associates a Servicing Thread with this Event Driven.
• DisableCompletionDetection (see page 765)	Disables the detection of request completion.
• DisableEventsDetection (see page 765)	Disables the detection of events.
• EnableCompletionDetection (see page 766)	Enables the detection of request completion.
• EnableEventsDetection (see page 766)	Enables the detection of events.
• FinalizeAndReleaseA (see page 766)	Finalizes and releases an Event Driven.
• GetIComUnknown (see page 766)	Returns a pointer to the Servicing Thread. AddIfRef is already called.
• GetServicingThread (see page 767)	Returns a pointer to the Servicing Thread. AddIfRef is already called.
• IsCurrentExecutionContext (see page 767)	Returns whether or not the code is executing within the current execution context.
• PostMessage (see page 767)	Pushes a new message into the message queue.
• RegisterRequestStatus (see page 768)	Registers a request status.
• RegisterSocket (see page 768)	Registers a socket.
• Release (see page 768)	Releases an Event Driven.

• StartTimer (see page 769)	Starts a new linear timer.
• StopAllTimers (see page 770)	Stops all timers owned by a manager.
• StopTimer (see page 770)	Stops a timer owned by a manager.
• UnregisterRequestStatus (see page 770)	Unregisters a request status.
• UnregisterRequestStatus (see page 770)	Unregisters a request status.
• UnregisterSocket (see page 771)	Unregisters a socket.

ITimerServiceMgr Class

ITimerServiceMgr Class	Description
• A EvTimerServiceMgrAwaken (see page 788)	Notifies the manager that a new timer elapsed or has been stopped.

ISocketServiceMgr Class

ISocketServiceMgr Class	Description
• A EvSocketServiceMgrAwaken (see page 785)	Notifies the manager about newly detected events on a socket.

IRequestStatusServiceMgr Class

IRequestStatusServiceMgr Class	Description
• A EvRequestStatusServiceMgrAwaken (see page 782)	Notifies the manager about newly completed request.

IMessageServiceMgr Class

IMessageServiceMgr Class	Description
• A EvMessageServiceMgrAwaken (see page 779)	Notifies the manager that a new message must be processed.

Legend

•	Method
▼	virtual
▲	abstract

2.18.1.1.1 - Constructors

2.18.1.1.1.1 - CSntpClient::CSntpClient Constructor

Constructor.

C++

```
CSntpClient();
```

Description

Constructor.

2.18.1.1.2 - Destructors

2.18.1.1.2.1 - CSntpClient::~CSntpClient Destructor

Destructor.

C++

```
virtual ~CSntpClient();
```

Description

Destructor.

2.18.1.1.3 - Methods

2.18.1.1.3.1 - CSntpClient::Activate Method

Activates the thread mechanism of the SNTP client.

C++

```
mxt_result Activate(IN uint32_t uStackSize = 0, IN CThread::EPriority ePriority = CThread::eNORMAL);
```

Parameters

Parameters	Description
IN uint32_t uStackSize = 0	The stack size associated with the SNTP client thread.
IN CThread::EPriority ePriority = CThread::eNORMAL	The priority associated with the SNTP client thread.

Returns

- resS_OK
- resFE_INVALID_STATE
- resFE_INVALID_ARGUMENT

Description

Activates the thread mechanism of the SNTP client.

2.18.1.1.3.2 - CSntpClient::ClearServerList Method

Clears the SNTP servers list.

C++

```
void ClearServerList();
```

Description

Clears the SNTP servers list.

2.18.1.1.3.3 - CSntpClient::GetTimeS Method

Gets the time.

C++

```
virtual mxt_result GetTimeS(OUT CTime& rTime);
```

Parameters

Parameters	Description
rTime	Structure used to return the time.

Returns

A mxt_result set to resS_OK or resFE_FAIL.

Description

Gets the time from the source.

Notes

This method is synchronized and may take some time to return.

2.18.1.1.3.4 - CSntpClient::InsertServer Method

Inserts a new server into the servers list.

C++

```
void InsertServer(IN const CFqdn& rServer);
```

Parameters

Parameters	Description
IN const CFqdn& rServer	The address of the server to add.

Description

Inserts a new server address.

2.18.1.1.3.5 - QueryServer

2.18.1.1.3.5.1 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(IN const CFqdn& rServer, OUT CTime& rTime, IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
IN const CFqdn& rServer	The address of the server where to send the SNTP request.
OUT CTime& rTime	A container for the result reception.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the server passed as parameter.

2.18.1.1.3.5.2 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(IN const CFqdn& rServer, OUT uint16_t& ruYear, OUT uint16_t& ruMonth, OUT uint16_t& ruDay, OUT uint16_t& ruHour, OUT uint16_t& ruMinute, OUT uint16_t& ruSecond, OUT uint16_t& ruMs, IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
IN const CFqdn& rServer	The address of the server where to send the SNTP request.
OUT uint16_t& ruYear	Year in numeric notation.
OUT uint16_t& ruMonth	Month in numeric notation.
OUT uint16_t& ruDay	Day in numeric notation.
OUT uint16_t& ruHour	Hour in numeric notation.
OUT uint16_t& ruMinute	Minutes in numeric notation.
OUT uint16_t& ruSecond	Seconds in numeric notation.
OUT uint16_t& ruMs	Milliseconds in numeric notation.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the server passed as parameter.

2.18.1.1.3.5.3 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(IN const CFqdn& rServer, OUT uint32_t& ruJulianDateDay, OUT uint32_t& ruJulianTimeMs, IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
IN const CFqdn& rServer	The address of the server where to send the SNTP request.
OUT uint32_t& ruJulianDateDay	Julian date day.

OUT uint32_t & ruJulianTimeMs	Julian time in ms.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the server passed as parameter.

2.18.1.1.3.5.4 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(OUT CTime& rTime, IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
OUT CTime& rTime	A container for the result reception.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request.

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the registered servers.

2.18.1.1.3.5.5 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(OUT uint16_t & ruYear, OUT uint16_t & ruMonth, OUT uint16_t & ruDay, OUT uint16_t & ruHour,
OUT uint16_t & ruMinute, OUT uint16_t & ruSecond, OUT uint16_t & ruMs, IN uint64_t uTimeoutMs =
uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
OUT uint16_t & ruYear	Year in numeric notation.
OUT uint16_t & ruMonth	Month in numeric notation.
OUT uint16_t & ruDay	Day in numeric notation.
OUT uint16_t & ruHour	Hour in numeric notation.
OUT uint16_t & ruMinute	Minutes in numeric notation.
OUT uint16_t & ruSecond	Seconds in numeric notation.
OUT uint16_t & ruMs	Milliseconds in numeric notation.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request.

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the registered servers.

2.18.1.1.3.5.6 - CSntpClient::QueryServer Method

Requests the current date and time from the registered servers.

C++

```
mxt_result QueryServer(OUT uint32_t & ruJulianDateDay, OUT uint32_t & ruJulianTimeMs, IN uint64_t uTimeoutMs =
uUSE_STATIC_CONFIGURATION);
```

Parameters

Parameters	Description
OUT uint32_t & ruJulianDateDay	Julian date day.
OUT uint32_t & ruJulianTimeMs	Julian time in ms.
IN uint64_t uTimeoutMs = uUSE_STATIC_CONFIGURATION	The timeout to the rejection of a SNTP request.

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Requests the current date and time from the registered servers.

2.18.1.1.3.6 - CSntpClient::SetSntpPacketSentToServer Method

Sets the number of packets sent to a server at each request.

C++

```
void SetSntpPacketSentToServer(IN uint16_t uSntpPacketSentToServer);
```

Parameters

Parameters	Description
IN uint16_t uSntpPacketSentToServer	Number of packets.

Description

Sets the number of packets sent to a server at each request.

2.18.1.1.3.7 - CSntpClient::SetTimeoutMs Method

Sets the default connection timeout.

C++

```
void SetTimeoutMs(IN uint64_t uTimeoutMs);
```

Parameters

Parameters	Description
IN uint64_t uTimeoutMs	Timeout in milliseconds.

Description

Sets the timeout, in ms, used when the CSntpClient (see page 816) user does not provide the timeout in ms when it calls the method QueryServer (see page 820).

2.18.1.1.3.8 - CSntpClient::Terminate Method

Terminates the thread mechanism of the SNTP client.

C++

```
void Terminate();
```

Description

Terminates the thread mechanism of the SNTP client. This method returns when the object is completely terminated.

2.18.1.2 - CTime Class

Class for retrieving the date and time.

Class Hierarchy

```
CTime
```

C++

```
class CTime;
```

Description

CTime returns the current date and time.

The default time when it is not possible to get the current date and time starts at 2000 January 1ST, time 0:0:0. Remember that this time might be readjusted if you set a time zone.

Published Interface:

Constructors

Constructor	Description
• CTime (see page 824)	Constructor.

Legend

◆	Method
---	--------

Destructors

Destructor	Description
• ~CTime (see page 824)	Destructor.

Legend

◆	Method
V	virtual

Operators

Operator	Description
• != (see page 830)	Different than operator.
• < (see page 830)	Less than operator.
• <= (see page 830)	Less than or equal to operator.
• = (see page 831)	Assignment operator.
• == (see page 831)	Comparison operator.
• > (see page 831)	Greater than operator.
• >= (see page 832)	Greater than or equal to.

Legend

◆	Method
---	--------

Methods

Method	Description
• GetDate (see page 824)	Gets the date.
• GetDateAndTime (see page 825)	Gets the date and time.
• GetDayOfWeek (see page 825)	Gets the day of week.
• GetGmtTime (see page 825)	Returns the time related to GMT in a tm structure.
• GetGregorianDate (see page 825)	Gets the date in Gregorian format.
• GetJulianDate (see page 826)	Gets the date in Julian format.
• GetJulianDateAndTime (see page 826)	Gets the Julian date and time.
• GetSystemTimeZone (see page 826)	Gets the system time zone.
• GetTime (see page 827)	Gets the time.
• IsDayLightSavingInEffect (see page 827)	Returns whether or not daylight saving time is in effect.
• IsLeapYear (see page 827)	Returns whether or not it is a leap year.
• PinTime (see page 827)	Pins the time in the current CTime.
• SetDate (see page 828)	Sets the date.
• SetDateAndTime (see page 828)	Sets the date and time.
• SetJulianDateAndTime (see page 828)	Sets the Julian date and time.
• SetSystemTime (see page 829)	Sets the system time using the OS time of day.
• SetSystemTimeZone (see page 829)	Sets the system time zone.
• SetTime (see page 829)	Sets the time.
• SetTimeZone (see page 830)	Sets the current location's time zone.

Legend

◆	Method
---	--------

2.18.1.2.1 - Constructors

2.18.1.2.1.1 - CTime

2.18.1.2.1.1.1 - CTime::CTime Constructor

Constructor.

C++

```
CTime();
```

Description

CTime contains the implementation of a real time clock. The jitter in time depends on the precision of the CTimer::GetSystemUpTimeMs (see page 503) method.

2.18.1.2.1.1.2 - CTime::CTime Constructor

Copy constructor.

C++

```
CTime(IN const CTime& rTime);
```

Parameters

Parameters	Description
IN const CTime& rTime	The CTime instance to init from.

Description

CTime contains the implementation of a real time clock. The jitter in time depends on the precision of the CTimer::GetSystemUpTimeMs (see page 503) method.

2.18.1.2.2 - Destructors

2.18.1.2.2.1 - CTime::~CTime Destructor

Destructor.

C++

```
virtual ~CTime();
```

Description

Destructor.

2.18.1.2.3 - Methods

2.18.1.2.3.1 - CTime::GetDate Method

Gets the date.

C++

```
bool GetDate(OUT uint16_t& ruYear, OUT uint16_t& ruMonth, OUT uint16_t& ruDay, IN bool bUTC = false) const;
```

Parameters

Parameters	Description
OUT uint16_t& ruYear	Year in numeric notation.
OUT uint16_t& ruMonth	Month in numeric notation.
OUT uint16_t& ruDay	Day in numeric notation.
IN bool bUTC = false	Indicates whether or not the date supplied is in UTC.

Returns

True if time is backed by an external source, false otherwise. This means that the base time is not based on the default time value.

Description

Gets the date from this instance of CTime (see page 822).

2.18.1.2.3.2 - CTime::GetDateAndTime Method

Gets the date and time.

C++

```
bool GetDateAndTime(OUT uint16_t& ruYear, OUT uint16_t& ruMonth, OUT uint16_t& ruDay, OUT uint16_t& ruHour, OUT
uint16_t& ruMinute, OUT uint16_t& ruSecond, OUT uint16_t& ruMs, IN bool bUTC = false) const;
```

Parameters

Parameters	Description
OUT uint16_t& ruYear	Year in numeric notation.
OUT uint16_t& ruMonth	Month in numeric notation.
OUT uint16_t& ruDay	Day in numeric notation.
OUT uint16_t& ruHour	Hours in numeric notation.
OUT uint16_t& ruMinute	Minutes in numeric notation.
OUT uint16_t& ruSecond	Seconds in numeric notation.
OUT uint16_t& ruMs	Milliseconds in numeric notation.
IN bool bUTC = false	Indicates whether or not the date and time should be in UTC.

Returns

True if time is backed by an external source, false otherwise. This means that the base time is not based on the default time value.

Description

Gets the date and time from this instance of CTime (see page 822).

2.18.1.2.3.3 - CTime::GetDayOfWeek Method

Gets the day of week.

C++

```
CTime::EDayOfWeek GetDayOfWeek(IN bool bUTC) const;
```

Parameters

Parameters	Description
IN bool bUTC	Indicates whether or not the day should be in UTC.

Returns

The current day of the week.

Description

Returns the current day of the week, SUNDAY being 0, MONDAY being 1, etc.

2.18.1.2.3.4 - CTime::GetGmtTime Method

Returns the time related to GMT in a tm structure.

C++

```
static bool GetGmtTime(OUT struct tm& rTime);
```

2.18.1.2.3.5 - CTime::GetGregorianDate Method

Gets the date in Gregorian format.

C++

```
static void GetGregorianDate(IN uint32_t uJulian, OUT uint16_t* puYear, OUT uint16_t* puMonth, OUT uint16_t*
```

```
puDay);
```

Parameters

Parameters	Description
IN uint32_t uJulian	Julian date to convert.
OUT uint16_t* puYear	Pointer to a uint16_t (see page 85) in which to put the Gregorian year.
OUT uint16_t* puMonth	Pointer to a uint16_t (see page 85) in which to put the Gregorian month.
OUT uint16_t* puDay	Pointer to a uint16_t (see page 85) in which to put the Gregorian day.

Description

Converts from a Julian date to a Gregorian date.

2.18.1.2.3.6 - CTime::GetJulianDate Method

Gets the date in Julian format.

C++

```
static uint32_t GetJulianDate(IN uint16_t uYear, IN uint16_t uMonth, IN uint16_t uDay);
```

Parameters

Parameters	Description
IN uint16_t uYear	Year in numeric notation.
IN uint16_t uMonth	Month in numeric notation.
IN uint16_t uDay	Day in numeric notation.

Returns

The converted Julian date.

Description

Converts from a Gregorian date to a Julian date.

2.18.1.2.3.7 - CTime::GetJulianDateAndTime Method

Gets the Julian date and time.

C++

```
bool GetJulianDateAndTime(OUT uint32_t& ruJulianDateDay, OUT uint32_t& ruJulianTimeMs, IN bool bUTC = false)  
const;
```

Parameters

Parameters	Description
OUT uint32_t& ruJulianDateDay	Julian date day.
OUT uint32_t& ruJulianTimeMs	Julian time in ms.
IN bool bUTC = false	Indicates whether or not the date and time should be in UTC.

Returns

True if time is backed by an external source, false otherwise. This means that the base time is not based on the default time value.

Description

Gets the Julian date and time from this instance of CTime (see page 822).

2.18.1.2.3.8 - CTime::GetSystemTimeZone Method

Gets the system time zone.

C++

```
static CString GetSystemTimeZone();
```

Returns

A CString (see page 126) that contains the system time zone string.

Description

Returns the Time Zone conversion string that is used when CTime (see page 822) has to convert from/to Locale to/from UTC. You should refer to CTimeZone.h for a description of the format.

2.18.1.2.3.9 - CTime::GetTime Method

Gets the time.

C++

```
bool GetTime(OUT uint16_t& ruHour, OUT uint16_t& ruMinute, OUT uint16_t& ruSecond, OUT uint16_t& ruMs, IN bool bUTC = false) const;
```

Parameters

Parameters	Description
OUT uint16_t& ruHour	Hours in numeric notation.
OUT uint16_t& ruMinute	Minutes in numeric notation.
OUT uint16_t& ruSecond	Seconds in numeric notation.
OUT uint16_t& ruMs	Milliseconds in numeric notation.
IN bool bUTC = false	Indicates whether or not the date and time should be in UTC.

Returns

True if time is backed by an external source, false otherwise. This means that the base time is not based on the default time value.

Description

Gets the time from this instance of CTime (see page 822).

2.18.1.2.3.10 - CTime::IsDayLightSavingInEffect Method

Returns whether or not daylight saving time is in effect.

C++

```
bool IsDayLightSavingInEffect() const;
```

Returns

True is it is in effect, false otherwise.

Description

Returns whether or not the current time is in the daylight saving period.

2.18.1.2.3.11 - CTime::IsLeapYear Method

Returns whether or not it is a leap year.

C++

```
bool IsLeapYear(IN bool bUTC = false) const;
```

Parameters

Parameters	Description
IN bool bUTC = false	Indicates whether or not the date and time should be in UTC.

Returns

True is it is a leap year, false otherwise.

Description

Returns whether or not the current year is a leap year.

2.18.1.2.3.12 - CTime::PinTime Method

Pins the time in the current CTime (see page 822).

C++

```
void PinTime();
```

Description

Pins the time in this CTime (see page 822).

2.18.1.2.3.13 - CTime::SetDate Method

Sets the date.

C++

```
void SetDate(IN uint16_t uYear, IN uint16_t uMonth, IN uint16_t uDay, IN bool bUTC = false);
```

Parameters

Parameters	Description
IN uint16_t uYear	Year in numeric notation.
IN uint16_t uMonth	Month in numeric notation.
IN uint16_t uDay	Day in numeric notation.
IN bool bUTC = false	Indicates whether or not the supplied date and time is in UTC.

Description

Sets the date to this instance of CTime (see page 822)

2.18.1.2.3.14 - CTime::SetDateAndTime Method

Sets the date and time.

C++

```
void SetDateAndTime(IN uint16_t uYear, IN uint16_t uMonth, IN uint16_t uDay, IN uint16_t uHour, IN uint16_t uMinute, IN uint16_t uSecond, IN uint16_t uMs, IN bool bUTC = false);
```

Parameters

Parameters	Description
IN uint16_t uYear	Year in numeric notation.
IN uint16_t uMonth	Month in numeric notation.
IN uint16_t uDay	Day in numeric notation.
IN uint16_t uHour	Hour in numeric notation.
IN uint16_t uMinute	Minutes in numeric notation.
IN uint16_t uSecond	Seconds in numeric notation.
IN uint16_t uMs	Milliseconds in numeric notation.
IN bool bUTC = false	Indicates whether or not the supplied date and time is in UTC.

Description

Sets the date and time to this instance of CTime (see page 822)

2.18.1.2.3.15 - CTime::SetJulianDateAndTime Method

Sets the Julian date and time.

C++

```
void SetJulianDateAndTime(IN uint32_t uJulianDateDay, IN uint32_t uJulianTimeMs, IN bool bUTC = false);
```

Parameters

Parameters	Description
IN uint32_t uJulianDateDay	Julian date day.
IN uint32_t uJulianTimeMs	Julian time in ms.
IN bool bUTC = false	Indicates whether or not the supplied date and time is in UTC.

Description

Sets the Julian date and time to this instance of CTime (see page 822)

2.18.1.2.3.16 - SetSystemTime

2.18.1.2.3.16.1 - CTime::SetSystemTime Method

Sets the system time using the OS time of day.

C++

```
static mxt_result SetSystemTime();
```

Description

Sets the system time using the OS time of day. Supported under LINUX only.

2.18.1.2.3.16.2 - CTime::SetSystemTime Method

Sets the system time.

C++

```
static void SetSystemTime(IN const CTime& rTime);
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the Ctime object to set.

Description

Sets the system time.

2.18.1.2.3.17 - CTime::SetSystemTimeZone Method

Sets the system time zone.

C++

```
static bool SetSystemTimeZone(IN const char* pszTimeZone);
```

Parameters

Parameters	Description
IN const char* pszTimeZone	Pointer to the buffer holding the time zone.

Returns

True is returned if no error, false otherwise.

Description

Specifies the Time Zone conversion string to use when CTime (see page 822) has to convert from/to Locale to/from UTC. You should refer to CTimeZone.h for a description of the format.

2.18.1.2.3.18 - CTime::SetTime Method

Sets the time.

C++

```
void SetTime(IN uint16_t uHour, IN uint16_t uMinute, IN uint16_t uSecond, IN uint16_t uMs, IN bool bUTC = false);
```

Parameters

Parameters	Description
IN uint16_t uHour	Hour in numeric notation.
IN uint16_t uMinute	Minutes in numeric notation.
IN uint16_t uSecond	Seconds in numeric notation.
IN uint16_t uMs	Milliseconds in numeric notation.
IN bool bUTC = false	Indicates whether or not the supplied date and time is in UTC.

Description

Sets the time to this instance of CTime (see page 822)

2.18.1.2.3.19 - CTime::SetTimeZone Method

Sets the current location's time zone.

C++

```
bool SetTimeZone(IN const char* pszTimeZone);
```

Parameters

Parameters	Description
IN const char* pszTimeZone	String representing the time zone.

Returns

True if no error.

Description

Sets the time zone for this instance of CTime (see page 822).

2.18.1.2.4 - Operators

2.18.1.2.4.1 - CTime::!= Operator

Different than operator.

C++

```
bool operator !=(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if both CTime (see page 822) objects are different, false otherwise.

Description

Checks whether or not both CTime (see page 822) objects are different to one another.

2.18.1.2.4.2 - CTime::< Operator

Less than operator.

C++

```
bool operator <(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if the left hand CTime (see page 822) is less than the right hand one, false otherwise.

Description

Checks whether or not the left hand CTime (see page 822) is less than the right hand one.

2.18.1.2.4.3 - CTime::<= Operator

Less than or equal to operator.

C++

```
bool operator <=(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if the left hand CTime (see page 822) is less than or equal to the right hand one, false otherwise.

Description

Checks whether or not the left hand CTime (see page 822) is less than or equal to the right hand one.

2.18.1.2.4.4 - CTime::= Operator

Assignment operator.

C++

```
CTime operator =(IN const CTime& rTime);
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime object to assign.

Returns

A reference to the assigned CTime (see page 822).

Description

Assigns the right hand CTime (see page 822) to the left hand one.

2.18.1.2.4.5 - CTime::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if both CTime (see page 822) objects are equal, false otherwise.

Description

Checks whether or not both CTime (see page 822) objects are equal to one another.

2.18.1.2.4.6 - CTime::> Operator

Greater than operator.

C++

```
bool operator >(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if the left hand CTime (see page 822) is greater than the right hand one, false otherwise.

Description

Checks whether or not the left hand CTime (see page 822) is greater than the right hand one.

2.18.1.2.4.7 - CTime::>= Operator

Greater than or equal to.

C++

```
bool operator >=(IN const CTime& rTime) const;
```

Parameters

Parameters	Description
IN const CTime& rTime	Reference to the CTime (see page 822) object to compare.

Returns

True if the left hand CTime (see page 822) is greater than or equal to the right hand one, false otherwise.

Description

Checks whether or not the left hand CTime (see page 822) is greater than or equal to the right hand one.

2.18.1.3 - CTimeZone Class

Implements the conversion between time zones and the handling of daylight saving time.

Class Hierarchy

```
CTimeZone
```

C++

```
class CTimeZone;
```

Description

```
class CTimeZone
```

CTimeZone does the conversion between UTC time/date and other time zones. It also takes care of the daylight saving time.

The CTimeZone needs to have the time zone string to do this conversion. This string follows the POSIX TZ string format standard.

STDOFFSET[DLST[OFFSET][,(Jn|M)[START[/TIME (see page 816)][,(J|M)[END[/TIME (see page 816)]]]]]

STD and DLST Three or more characters that are the designation for the standard (STD) or alternate daylight saving time (DLST) time zone. Only STD is required. If DLST is not supplied, the daylight saving time does not apply to the locale. Lower and upper case letters are allowed. Any characters, except digits, leading colon (:), comma (,), minus (-), plus (+), and ASCII NUL, are allowed.

OFFSET Indicates the value to be added to the local time to arrive at UTC. The offset has the format h[h]:m[m]:s[s]]. If no offset is supplied for DLST, the alternate time is assumed to be one hour ahead of standard time. One or more digits can be used; the value is always interpreted as a decimal number. The hour value must be between 0 and 24. The minute and seconds values, if present, must be between 0 and 59. If preceded by a minus (-), the time zone is east of the prime meridian, otherwise it is west, which can be indicated by the preceding plus sign (+)

START and END Indicates when to change to and return from the daylight saving time. The START argument is the date when the change from the standard to the daylight saving time occurs; the END argument is the date for changing back. If START and END are not specified, the default is the US Daylight saving time start and end dates. The format for start and end must be one of the following:

n, where n is the number of days since the start of the year from 0 to 365. This n must contain the leap year day if the current year is a leap year. With this format, it is the user's responsibility to determine all the leap year details. This was not really shown in the time zone string format at the top of the header because it was simpler for the user. If you look, you will see that Jn is optional; when Jn is not there, the current case applies. It is possible to refer to February 29.

Jn, where n is the number of days since the start of the year from 1 to 365. Leap days are not counted. The TIME (see page 816) parameter has the same format as OFFSET but there can be no leading minus (-) or plus (+) sign. If TIME (see page 816) is not specified, the default is 02:00:00. For example, February 28 is day 59, March 1 is day 60. It is impossible to refer to the occasional February 29 explicitly.

Mx[x].y.z, where x is the month, y is a week count and z is the day of the week starting at 0 (Sunday). When y is 5, it refers to the last z day of month x which may occur in either the fourth or fifth week. Week 1 is the first week in which the z day occurs. Day 0 is Sunday. As an example, M10.5.0 is the Sunday of the fifth week of October. As in Jn, the parameter may also be supplied. If not supplied, the default is 02:00:00. 1 <= x <= 12, 1 <= y <= 5, 0 <= z <= 6

Published Interface:

Constructors

Constructor	Description
 CTimeZone (see page 833)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
 ~CTimeZone (see page 834)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 837)	Assignment operator.

Legend

	Method
---	--------

Methods

Method	Description
 ConvertFromLocaleToUTC (see page 834)	Converts the time from local to UTC.
 ConvertFromUTCToLocale (see page 835)	Converts the time from UTC to local.
 GetTimeZone (see page 835)	Gets the time zone.
 IsDayLightSavingInEffect (see page 836)	Verifies if the daylight saving time is in effect.
 IsLeapYear (see page 836)	Verifies if the supplied year is a leap year.
 IsValid (see page 837)	Verifies if the time zone is valid.
 SetTimeZone (see page 837)	Sets the time zone.

Legend

	Method
---	--------

2.18.1.3.1 - Constructors

2.18.1.3.1.1 - CTimeZone

2.18.1.3.1.1.1 - CTimeZone::CTimeZone Constructor

Constructor.

C++

```
CTimeZone();
```

Description

Constructor.

2.18.1.3.1.1.2 - CTimeZone::CTimeZone Constructor

Copy constructor.

C++

```
CTimeZone(IN const CTimeZone& rTimeZone);
```

Parameters

Parameters	Description
IN const CTimeZone& rTimeZone	A reference to the time zone to copy.

Description

Copy constructor.

2.18.1.3.1.1.3 - CTimeZone::CTimeZone Constructor

Constructor. Builds time zone using the string.

C++

```
CTimeZone( IN const char* pszTimeZone );
```

Parameters

Parameters	Description
IN const char* pszTimeZone	Pointer to the time zone to set.

Description

Constructor. Creates the time zone by using the specified string.

2.18.1.3.2 - Destructors**2.18.1.3.2.1 - CTimeZone::~CTimeZone Destructor**

Destrcutor.

C++

```
virtual ~CTimeZone();
```

Description

Destructor.

2.18.1.3.3 - Methods**2.18.1.3.3.1 - ConvertFromLocaleToUTC****2.18.1.3.3.1.1 - CTimeZone::ConvertFromLocaleToUTC Method**

Converts the time from local to UTC.

C++

```
void ConvertFromLocaleToUTC( IN OUT uint16_t& ruYear, IN OUT uint16_t& ruMonth, IN OUT uint16_t& ruDay, IN OUT
uint16_t& ruHour, IN OUT uint16_t& ruMinute, IN OUT uint16_t& ruSecond, IN OUT uint16_t& ruMs ) const;
```

Parameters

Parameters	Description
IN OUT uint16_t& ruYear	Year in numeric notation.
IN OUT uint16_t& ruMonth	Month in numeric notation.
IN OUT uint16_t& ruDay	Day in numeric notation.
IN OUT uint16_t& ruHour	Hours in numeric notation.
IN OUT uint16_t& ruMinute	Minutes in numeric notation.
IN OUT uint16_t& ruSecond	Seconds in numeric notation.
IN OUT uint16_t& ruMs	Milliseconds in numeric notation.

Description

Converts from a local date and time to a UTC date and time.

2.18.1.3.3.1.2 - CTimeZone::ConvertFromLocaleToUTC Method

Converts the time from local to UTC.

C++

```
void ConvertFromLocaleToUTC(IN OUT uint32_t& ruJulianDateDay, IN OUT uint32_t& ruJulianTimeMs) const;
```

Parameters

Parameters	Description
IN OUT uint32_t& ruJulianDateDay	Julian date day.
IN OUT uint32_t& ruJulianTimeMs	Julian time in ms.

Description

Converts from a local date and time to a UTC date and time.

2.18.1.3.3.2 - CTimeZone::ConvertFromUTCToLocale**2.18.1.3.3.2.1 - CTimeZone::ConvertFromUTCToLocale Method**

Converts the time from UTC to local.

C++

```
void ConvertFromUTCToLocale(IN OUT uint16_t& ruYear, IN OUT uint16_t& ruMonth, IN OUT uint16_t& ruDay, IN OUT
uint16_t& ruHour, IN OUT uint16_t& ruMinute, IN OUT uint16_t& ruSecond, IN OUT uint16_t& ruMs) const;
```

Parameters

Parameters	Description
IN OUT uint16_t& ruYear	Year in numeric notation.
IN OUT uint16_t& ruMonth	Month in numeric notation.
IN OUT uint16_t& ruDay	Day in numeric notation.
IN OUT uint16_t& ruHour	Hours in numeric notation.
IN OUT uint16_t& ruMinute	Minutes in numeric notation.
IN OUT uint16_t& ruSecond	Seconds in numeric notation.
IN OUT uint16_t& ruMs	Milliseconds in numeric notation.

Description

Converts from a UTC date and time to a local date and time.

2.18.1.3.3.2.2 - CTimeZone::ConvertFromUTCToLocale Method

Converts the time from UTC to local.

C++

```
void ConvertFromUTCToLocale(IN OUT uint32_t& ruJulianDateDay, IN OUT uint32_t& ruJulianTimeMs) const;
```

Parameters

Parameters	Description
IN OUT uint32_t& ruJulianDateDay	Julian date day.
IN OUT uint32_t& ruJulianTimeMs	Julian time in ms.

Description

Converts from a UTC date and time to a local date and time

2.18.1.3.3.3 - CTimeZone::GetTimeZone Method

Gets the time zone.

C++

```
const char* GetTimeZone() const;
```

Returns

The time zone

Description

Gets the time zone stored in this CTimeZone (see page 832) object.

2.18.1.3.3.4 - IsDayLightSavingInEffect**2.18.1.3.3.4.1 - CTimeZone::IsDayLightSavingInEffect Method**

Verifies if the daylight saving time is in effect.

C++

```
bool IsDayLightSavingInEffect(IN uint16_t uYear, IN uint16_t uMonth, IN uint16_t uDay, IN uint16_t uHour, IN
uint16_t uMinute, IN uint16_t uSecond, IN uint16_t uMs, IN bool bUTC) const;
```

Parameters

Parameters	Description
IN uint16_t uYear	Year in numeric notation.
IN uint16_t uMonth	Month in numeric notation.
IN uint16_t uDay	Day in numeric notation.
IN uint16_t uHour	Hours in numeric notation.
IN uint16_t uMinute	Minutes in numeric notation.
IN uint16_t uSecond	Seconds in numeric notation.
IN uint16_t uMs	Milliseconds in numeric notation.
IN bool bUTC	Indicates whether or not the date and time should be in UTC.

Returns

True if daylight saving time is in effect, false otherwise.

Description

Returns true if the date and time passed by parameter is located in the daylight saving period

2.18.1.3.3.4.2 - CTimeZone::IsDayLightSavingInEffect Method

Verifies if the daylight saving time is in effect.

C++

```
bool IsDayLightSavingInEffect(IN uint32_t uJulianDateDay, IN uint32_t uJulianTimeMs, IN bool bUTC) const;
```

Parameters

Parameters	Description
IN uint32_t uJulianDateDay	Julian date day.
IN uint32_t uJulianTimeMs	Julian time in ms.
IN bool bUTC	True if the date and time passed are in UTC, false otherwise.

Returns

True if daylight saving time is in effect, false otherwise.

Description

Returns true if the date and time passed as parameter is located in the daylight saving period. The flag bUTC is set to true when the passed parameters are UTC related.

2.18.1.3.3.5 - CTimeZone::IsLeapYear Method

Verifies if the supplied year is a leap year.

C++

```
static bool IsLeapYear(IN uint16_t uYear);
```

Parameters

Parameters	Description
IN uint16_t uYear	Year to check. In YYYY format.

Returns

True if the year is a leap year, false otherwise.

Description

Returns whether or not the current year is a leap year.

2.18.1.3.3.6 - CTimeZone::IsValid Method

Verifies if the time zone is valid.

C++

```
bool IsValid();
```

Returns

True if the time zone is valid, false otherwise.

Description

Verifies if the time zone is valid.

2.18.1.3.3.7 - CTimeZone::SetTimeZone Method

Sets the time zone.

C++

```
mxt_result SetTimeZone( IN const char* pszTimeZone );
```

Parameters

Parameters	Description
IN const char* pszTimeZone	Time zone to set.

Returns

- resFE_INVALID_ARGUMENT
- resS_OK

Description

Sets the time zone. The user is required to call this method before calling the conversion methods. This method returns an error if the time zone is not in a valid format. See the CTimeZone.h header documentation for more information.

2.18.1.3.4 - Operators**2.18.1.3.4.1 - CTimeZone::= Operator**

Assignment operator.

C++

```
CTimeZone operator =( IN const CTimeZone& rTimeZone );
```

Parameters

Parameters	Description
IN const CTimeZone& rTimeZone	A reference to the time zone to copy.

Returns

A reference to the assigned CTimeZone (see page 832)

Description

Assigns the right hand CTimeZone (see page 832) to the left hand one.

2.18.1.4 - IExternalTimeSource Class

Base class to be inherited by every class that wants to be a time source.

Class Hierarchy

IExternalTimeSource

C++

```
class IExternalTimeSource;
```

Description

class IExternalTimeSource

IExternalTimeSource is a base class that must be inherited by every class that wants to become a time source, e.g., CSntpClient (see page 816).

Methods

Method	Description
◆ A GetTimeS (see page 838)	Gets the time.

Legend

◆	Method
◆ A	abstract

2.18.1.4.1 - Methods

2.18.1.4.1.1 - IExternalTimeSource::GetTimeS Method

Gets the time.

C++

```
virtual mxt_result GetTimeS(OUT CTime& rTime) = 0;
```

Parameters

Parameters	Description
OUT CTime& rTime	Structure used to return the time.

Returns

A mxt_result (see page 92) set to resS_OK or resFE_FAIL.

Description

Gets the time from the source.

Notes

This method is synchronized and may take some time to return.

2.18.2 - Enumerations

This section documents the enumerations of the Sources/Time folder.

Enumerations

Enumeration	Description
EDayOfWeek (see page 838)	Enumeration with the days of the week.

2.18.2.1 - CTime::EDayOfWeek Enumeration

Enumeration with the days of the week.

C++

```
enum EDayOfWeek {
    eSUNDAY,
    eMONDAY,
    eTUESDAY,
    eWEDNESDAY,
    eTHURSDAY,
    eFRIDAY,
    eSATURDAY
}
```

};

Members

Members	Description
eSUNDAY	Sunday.
eMONDAY	Monday.
eTUESDAY	Tuesday.
eWEDNESDAY	Wednesday
eTHURSDAY	Thursday
eFRIDAY	Friday.
eSATURDAY	Saturday.

2.18.3 - Variables

This section documents the variables of the Sources/Time folder.

2.18.3.1 - uJAN_1ST_1900 Variable

Julian representation of January 1st, 1900.

C++

```
const uint32_t uJAN_1ST_1900 = 2415021;
```

2.18.3.2 - uJAN_1ST_2000 Variable

Julian representation of January 1st, 2000.

C++

```
const uint32_t uJAN_1ST_2000 = 2451545;
```

2.18.3.3 - uMS_PER_DAY Variable

Number of milliseconds per day.

C++

```
const uint32_t uMS_PER_DAY = (24 * 60 * 60 * 1000);
```

2.18.3.4 - uSEC_PER_DAY Variable

Number of seconds per day.

C++

```
const uint32_t uSEC_PER_DAY = (24 * 60 * 60);
```

2.19 - Xml

This section documents the Sources/Xml folder of the M5T Framework. It is divided in functional subsections:

- Classes (see page 839)
- Enumerations (see page 883)

2.19.1 - Classes

This section documents the classes of the Sources/Xml folder.

Classes

Class	Description
CXmlElement (see page 840)	
CXmlParser (see page 867)	Implements a XML (see page 839) parser.
CXmlWriter (see page 869)	Implements methods for writing an XML (see page 839) document.
I XmlDocument (see page 874)	
I XmlParserHandler (see page 881)	Provides the interface for any class that wants to be an XML (see page 839) parser handler.
I XmlWriterOutputHandler (see page 882)	This file provides the interface for any class that wants to be an XmlWriter Manager.

2.19.1.1 - CXMLElement Class

Class Hierarchy

```
CXMLElement
```

C++

```
class CXMLElement;
```

Description

The CXMLElement represents an XML (see page 839) element. An XML (see page 839) element is best illustrated by an example:

```
<elem1>
  <elem2>
    TEST
  </elem2>
</elem1>
```

In the above example, both elem1 and elem2 are XML (see page 839) elements. XML (see page 839) elements always have a name. They can be associated with a namespace or not. They also can have attributes and values.

The XML (see page 839) element are structured as a tree to represent an XML (see page 839) document.

XML (see page 839) elements are always created either using an IXmlDocument (see page 874) or using another XML (see page 839) element.

Location

Xml/CXMLElement.h

See Also

[IXmlDocument](#) (see page 874)

Methods

Method	Description
AppendAttribute (see page 843)	Adds an attribute at the end of the current attribute list.
Copy (see page 843)	Copies an element into this element.
CreateChildElement (see page 844)	Creates a child element that is a copy of rFrom.
CreateElement (see page 848)	Creates a child or sibling element that is a copy of another element..
DeclareNamespace (see page 851)	Declares a namespace in this element.
Delete (see page 852)	Deletes this element.
DeleteAttribute (see page 852)	Deletes the attribute that has the specified name.
DeleteDeclaredNamespace (see page 853)	Deletes a declared namespace.
FindChildElement (see page 853)	Finds a child element according to its name and index.
GetAttribute (see page 854)	Gets an attribute that has the specified name.
GetChildElement (see page 854)	Gets a child element at the specified index.
GetDeclaredNamespaces (see page 855)	Gets the namespaces declared at this level.
GetFirstSibling (see page 855)	Gets the first sibling element from this element.
GetLastSibling (see page 856)	Gets the last sibling element from this element.
GetName (see page 856)	Gets the name of this element.
GetNamespace (see page 856)	Gets the namespace for this element.
GetNamespaceByPrefix (see page 857)	Gets the namespace for this element from the namespace prefix.
GetNamespaceByUri (see page 857)	Gets the namespace for this element from the namespace URI.
GetNamespacePrefix (see page 857)	Gets the prefix of the namespace for this element.
GetNextSibling (see page 858)	Gets the next sibling element to this element.
GetNumAttributes (see page 858)	Gets the number of attributes held by this element.
GetNumChildElement (see page 858)	Gets the number of child elements held by this element.
GetOpaque (see page 858)	Retrieves the opaque identifier.
GetParentElement (see page 859)	Gets the parent element to this element.
GetPreviousSibling (see page 859)	Gets the previous sibling element to this element.
GetValue (see page 860)	Gets the string value associated with this element.
GetXmlDocument (see page 860)	Returns the IXmlDocument (see page 874) value.
Serialize (see page 860)	Serializes the current element and children into a blob.
SetAttribute (see page 860)	Sets the value of the attribute or creates it if it does not exist.
SetChildElement (see page 861)	Creates or replaces a child element that is inserted following a specified order.

• SetName (see page 864)	Sets the name of this element.
• SetNamespace (see page 864)	Sets the namespace for this element.
• SetOpaque (see page 864)	Sets the opaque for this CXmlElement.
• SetValue (see page 865)	Sets the string value associated with this element.
• UpdateAttribute (see page 865)	Changes the namespace, name and value associated with an existing attribute.
• UpdateDeclaredNamespace (see page 865)	Updates the URI of a declared namespace.

Legend

	Method
---	--------

Enumerations

Enumeration	Description
ECreatePosition (see page 866)	
ENsDeclarationBehavior (see page 866)	
ENsDeclarationPosition (see page 867)	

Structs

Struct	Description
SIdentificationInfo (see page 841)	
SNamespace (see page 842)	Structure used to hold a chained list of namespaces.

2.19.1.1.1 - Structs

2.19.1.1.1.1 - CXmlElement::SIdentificationInfo Struct

Class Hierarchy

C++

```
struct SIdentificationInfo {
    const char* m_pszNamespace;
    const char* m_pszName;
};
```

Description

This structure permits identification of an XML (see page 839) component.

Constructors

Constructor	Description
• SIdentificationInfo (see page 842)	

Legend

	Method
---	--------

2.19.1.1.1.1.1 - Data Members

2.19.1.1.1.1.1.1 - CXmlElement::SIdentificationInfo::m_pszName Data Member

Description.

C++

```
const char* m_pszName;
```

Description

The name for which to search.

2.19.1.1.1.1.1.2 - CXmlElement::SIdentificationInfo::m_pszNamespace Data Member

```
const char* m_pszNamespace;
```

Description

The namespace where the name is defined.

2.19.1.1.1.1.2 - Constructors

2.19.1.1.1.1.2.1 - CXMLElement::SIdentificationInfo::SIdentificationInfo Constructor

```
SIdentificationInfo();
```

Description

Default constructor.

2.19.1.1.1.2 - CXMLElement::SNamespace Struct

Structure used to hold a chained list of namespaces.

Class Hierarchy

C++

```
struct SNamespace {
    enum ENamespaceCharacteristics {
        eURI_MUST_BE_DELETED = 0x01
    };
    char m_cCharacteristics;
    const char* m_pszUri;
    char* m_pszPrefix;
    SNamespace* m_pstNextNamespace;
};
```

Constructors

Constructor	Description
 SNamespace (See page 843)	

Legend

	Method
---	--------

Enumerations

Enumeration	Description
ENamespaceCharacteristics (See page 843)	

2.19.1.1.1.2.1 - Data Members

2.19.1.1.1.2.1.1 - CXMLElement::SNamespace::m_cCharacteristics Data Member

```
char m_cCharacteristics;
```

Description

A bitset keeping characteristics of this instance. This is a bitset of ENamespaceCharacteristics (See page 843) values.

2.19.1.1.1.2.1.2 - CXMLElement::SNamespace::m_pstNextNamespace Data Member

```
SNamespace* m_pstNextNamespace;
```

Description

Next namespace in the chained list.

2.19.1.1.1.2.1.3 - CXMLElement::SNamespace::m_pszPrefix Data Member

```
char* m_pszPrefix;
```

Description

The prefix string.

2.19.1.1.1.2.1.4 - CXMLElement::SNamespace::m_pszUri Data Member

```
const char* m_pszUri;
```

Description

The URI string.

2.19.1.1.1.2.2.2 - Constructors**2.19.1.1.1.2.2.2.1 - SNamespace****2.19.1.1.1.2.2.2.1.1 - CXMLElement::SNamespace::SNamespace Constructor**

```
SNamespace();
```

Description

Default constructor.

2.19.1.1.1.2.2.2.1.2 - CXMLElement::SNamespace::SNamespace Constructor

```
SNamespace(IN char cCharacteristics, IN const char* pszUri, IN char* pszPrefix, IN SNamespace* pstNextNamespace);
```

Description

Constructor.

2.19.1.1.1.2.3 - Enumerations**2.19.1.1.1.2.3.1 - CXMLElement::SNamespace::ENamespaceCharacteristics Enumeration**

```
enum ENamespaceCharacteristics {
    eURI_MUST_BE_DELETED = 0x01
};
```

Description

This enumeration is used to specify the namespace characteristics.

Members

Members	Description
eURI_MUST_BE_DELETED = 0x01	When this bitmask is present, m_pszUri (see page 842) was allocated with the IXmlDocument (see page 874) and must be released with it.

2.19.1.1.2 - Methods**2.19.1.1.2.1 - CXMLElement::AppendAttribute Method**

Adds an attribute at the end of the current attribute list.

C++

```
mxt_result AppendAttribute(IN const char* pszAttrNamespace, IN const char* pszAttrName, IN const char* pszAttrValue);
```

Parameters

Parameters	Description
IN const char* pszAttrNamespace	The namespace of the attribute.
IN const char* pszAttrName	The name of the attribute. It must not be NULL.
IN const char* pszAttrValue	The value of the attribute. It must not be NULL.

Description

This method appends an attribute to the end of the attribute list of the current element.

2.19.1.1.2.2 - CXMLElement::Copy Method

Copies an element into this element.

C++

```
mxt_result Copy(IN const CXMLElement& rFrom, IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const CXMLElement& rFrom	The CXMLElement (see page 840) to copy.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	When and how to create namespaces declared in rFrom and its children. The accepted values of ENsDeclarationBehavior (see page 866) are: <ul style="list-style-type: none"> • eNSDECLARE_FORCE: Forces the declaration of the namespaces found in rFrom and its children. This value can only be used with ePosition set to eNSPOS_AS_COPY. This has the effect of creating an exact copy of rFrom and its children, including all namespace declarations and their exact prefixes. • eNSDECLARE_OPTIMIZE: Declares the namespaces declared in rFrom only when needed. Moreover, the namespaces are declared in such a way as to avoid namespace prefix clashes.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	Where to declare the namespaces that are declared in rFrom. eNSPOS_NEW_ELEMENT can never be used when calling this Copy operation, as it is possible that rFrom has multiple children elements that would be considered as new elements in this operation, and it would be impossible to know in which new element to declare namespaces. eNSPOS_AS_COPY can only be used in conjunction with eNSDECLARE_FORCE, while the other allowed positions are possible with eNSDECLARE_OPTIMIZE.

Returns

resS_OK if the element has been copied, an error otherwise.

Description

This method copies the specified CXMLElement (see page 840) into this CXMLElement (see page 840).

The following is copied from rFrom:

- Name
- Value
- Opaque
- Declared namespace (as per eBehavior and ePosition)
- Attributes
- Child elements

Note:

The opaque of rFrom is also copied. If the caller does not want the opaque to change, it then MUST call SetOpaque (see page 864) after the call to Copy providing the old opaque.

2.19.1.1.2.3 - CreateChildElement**2.19.1.1.2.3.1 - CXMLElement::CreateChildElement Method**

Creates a child element that is a copy of rFrom.

C++

```
CXMLElement* CreateChildElement(IN const CXMLElement& rFrom, IN const CVector<SIdentificationInfo>* pvecstElementOrder, IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const CXMLElement& rFrom	Reference to the CXMLElement (see page 840) to copy.
IN const CVector<SIIdentificationInfo>* pvecstElementOrder	A pointer to a CVector (see page 227) giving the order of the elements. An element is added at the place its name/namespace are in the CVector (see page 227). To permit undefined names to be present in the children, insert an 'any' SIIdentificationInfo (see page 841) structure (i.e. with a NULL name and a NULL namespace). Note that if many 'any' SIIdentificationInfo (see page 841) are added, only the last one is taken into account. It can be NULL. In this case, the any element is inserted as the last child.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	When and how to create namespaces declared in rFrom and its children. The accepted values of ENsDeclarationBehavior (see page 866) are: <ul style="list-style-type: none"> • eNSDECLARE_FORCE: Forces the declaration of the namespaces found in rFrom and its children. This value can only be used with ePosition set to eNSPOS_AS_COPY. This has the effect of creating an exact copy of rFrom and its children, including all namespace declarations and their exact prefixes. • eNSDECLARE_OPTIMIZE: Declares the namespaces declared in rFrom only when needed. Moreover, the namespaces are declared in such a way as to avoid namespace prefix clashes.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	Where to declare the namespaces that are declared in rFrom. eNSPOS_AS_COPY can only be used in conjunction with eNSDECLARE_FORCE, while the other positions are possible with eNSDECLARE_OPTIMIZE.

Returns

The element that was created.

NULL if an error occurred.

Description

This method creates a child element that is a copy of another element.

The element is created and inserted at the place where it is in pvecstElementOrder. If an element with the same name and namespace is already present, the new one is inserted as the next element of the last element with the same name and namespace.

This method manages the child as if there is a sequence in which each element can have unlimited occurrences.

See Also

CreateChildElement

2.19.1.1.2.3.2 - CXmIElement::CreateChildElement Method

Creates a child element that is inserted following a specified order.

C++

```
CXMLElement* CreateChildElement(IN const char* pszNamespace, IN const char* pszNamespacePrefix, IN const char* pszElementName, IN const char* pszNewValue, IN const CVector<SIIdentificationInfo>* pvecstElementOrder, IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace of the element to create. pszNamespace MUST be present in pvecstElementOrder along with pszElementName UNLESS there is an 'any' element (i.e. {NULL, NULL}) in pvecstElementOrder. In the latter case, the element is searched after the element preceding the 'any' element and before the element following the 'any' element in pvecstElementOrder. The element is then inserted as the previous sibling of the element following the 'any' element. When creating an element that is already present, it is inserted after the last instance of the element. It MUST NOT be NULL.

IN const char* pszNamespacePrefix	The prefix for pszNamespaceUri. It can be NULL if pszNamespace is the default namespace. This parameter is ignored if pszNamespace is already declared in one of the parents and it must not be redeclared.
IN const char* pszElementName	The name of the element to create. pszElementName MUST be present in pvecstElementOrder along with pszNamespace UNLESS there is an 'any' element (i.e. {NULL, NULL}) in pvecstElementOrder. In the latter case, the element is searched after the element preceeding the 'any' element and before the element following the 'any' element in pvecstElementOrder. The element is inserted as the previous sibling of the element following the 'any' element. It MUST NOT be NULL.
IN const char* pszNewValue	The value to set in the created element. It can be NULL.
IN const CVector<SIIdentificationInfo>* pvecstElementOrder	A pointer to a CVector (see page 227) giving the order of the elements. An element is added at the place its name/namespace are in the CVector (see page 227). To permit undefined names to be present in the children, insert an 'any' SIIdentificationInfo (see page 841) structure (i.e. with a NULL name and a NULL namespace). Note that if many 'any' SIIdentificationInfo (see page 841) are added, only the last one is taken into account. It can be NULL. In this case, any element is inserted as the last child.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	The behavior to have to declare the namespace:
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	The position specifying where to declare missing namespaces.

Returns

The element that was created.

NULL if an error occurred.

Description

This method creates the element specified by pszNamespace and pszElementName.

The element is created and inserted at the place where it is in pvecstElementOrder, and its value is set to pszNewValue. If an element with the same name and namespace is already present, the new one is inserted as the next element of the last element with the same name and namespace.

This method manages the child as if there is a sequence in which each element can have unlimited occurrences.

See Also

SetChildElement (see page 861), CreateElement (see page 848), ENsDeclarationBehavior (see page 866), ENsDeclarationPosition (see page 867)

Example

This shows how the pvecstElementOrder works.

```
CVector<SIIdentificationInfo> vecstElementOrder;

// vecstElementOrder is initialized to contain
// {"urn:ietf:params:xml:ns:pidf:rpid", "note"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "audio"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "video"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "text"}
(...)

// pXmlElement is a new element containing no child element yet.
CXmlElement* pXmlElement = (...);

CXmlElement* pAudio1 = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "audio",
                                                       "value1",
                                                       &vecstElementOrder);

// pAudio1 is now set to a newly created element with value of "value1".

// pXmlElement now has the following children elements
// <rpid:audio>value1
//
// Where rpid is the in-scope namespace prefix of
// "urn:ietf:params:xml:ns:pidf:rpid"

CXmlElement* pNotel = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "note",
```

```

        "note1",
        &vecstElementOrder);

// pNote1 is now set to a newly created element with value of "note1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value1

CXmlElement* pText1 = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "text",
                                                       "text1",
                                                       &vecstElementOrder);

// pText1 is now set to a newly created element with value of "text1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value1
//  <rpid:text>text1

CXmlElement* pAudio2 = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "audio",
                                                       "value2",
                                                       &vecstElementOrder);

// pAudio2 is a new object created with a value of "value2".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value1
//  <rpid:audio>value2
//  <rpid:text>text1

CXmlElement* pVideo1 = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "video",
                                                       "video1",
                                                       &vecstElementOrder);

// pVideo1 is now set to a newly created element with value of "video1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value1
//  <rpid:audio>value2
//  <rpid:video>video1
//  <rpid:text>text1

```

This shows how to use the 'any' element:

```

CVector<SIdentificationInfo> vecstElementOrder;

// vecstElementOrder is initialized to contain
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "note"},
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "audio"},
//  {NULL,                                     NULL}, // any element
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "video"},
//  {"other:namespace",                      "note"}
(...)

// pXmlElement is an element containing the following children:
//  <rpid:note>note1
//  <rpid:audio>value2
//  <ons:note>endnote1
//
// Where rpid is the in-scope namespace prefix of
// "urn:ietf:params:xml:ns:pidf:rpid"
// and ons is the in-scope namespace prefix of "other:namespace".
CXmlElement* pXmlElement = (...);

CXmlElement* pCustom1 = pXmlElement->CreateChildElement("other:namespace",
                                                       "oth",
                                                       "custom",
                                                       "value1",
                                                       &vecstElementOrder);

// pCustom1 is now set to a newly created element with value of "value1".

// pXmlElement now has the following children elements

```

```

// <rpid:note>note1
// <rpid:audio>value2
// <oth:custom>value1
// <ons:note>endnote1

CXMLElement* pOther1 = pXmlElement->CreateChildElement("other:namespace",
                                                       "oth",
                                                       "other",
                                                       "othervalue1",
                                                       &vecstElementOrder);

// pOther1 is now set to a newly created element with value of "othervalue1".

// pXmlElement now has the following children elements
// <rpid:note>note1
// <rpid:audio>value2
// <oth:custom>value1
// <oth:other>othervalue1
// <ons:note>endnote1

CXMLElement* pCustom2 = pXmlElement->CreateChildElement("other:namespace",
                                                       "oth",
                                                       "custom",
                                                       "value2",
                                                       &vecstElementOrder);

// pCustom2 is now pointing to a new object set with the value "value2".

// pXmlElement now has the following children elements
// <rpid:note>note1
// <rpid:audio>value2
// <oth:custom>value1
// <oth:other>othervalue1
// <oth:custom>value2
// <ons:note>endnote1

CXMLElement* pVideo1 = pXmlElement->CreateChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                       "rpid",
                                                       "video",
                                                       "video1",
                                                       &vecstElementOrder);

// pVideo1 is now set to a newly created element with value of "video1".

// pXmlElement now has the following children elements
// <rpid:note>note1
// <rpid:audio>value2
// <oth:custom>value1
// <oth:other>othervalue1
// <oth:custom>value2
// <rpid:video>video1
// <ons:note>endnote1

```

2.19.1.1.2.4 - CreateElement

2.19.1.1.2.4.1 - CXMLElement::CreateElement Method

Creates a child or sibling element that is a copy of another element..

C++

```
CXMLElement* CreateElement(IN const CXMLElement& rFrom, IN ECreatePosition ePos, IN ENsDeclarationBehavior
eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const CXMLElement& rFrom	Reference to the CXMLElement (see page 840) to copy.

IN ECreatePosition ePos	The position where the element must be inserted.
	<ul style="list-style-type: none"> • ePOS_FIRST_SIBLING: The new element is created as the first child of the parent element. • ePOS_PREVIOUS_SIBLING: The new element is created and inserted as the previous sibling of this element. • ePOS_NEXT_SIBLING: The new element is created and inserted as the next sibling of this element. • ePOS_LAST_SIBLING: The new element is created and inserted as the last child of the parent element. • ePOS_FIRST_CHILD: The new element is created and inserted as the first child of this element. • ePOS_LAST_CHILD: The new element is created and inserted as the last child of this element.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	When and how to create namespaces declared in rFrom and its children. The accepted values of ENsDeclarationBehavior (see page 866) are: <ul style="list-style-type: none"> • eNSDECLARE_FORCE: Forces the declaration of the namespaces found in rFrom and its children. This value can only be used with ePosition set to eNSPOS_AS_COPY. This has the effect of creating an exact copy of rFrom and its children, including all namespace declarations and their exact prefixes. • eNSDECLARE_OPTIMIZE: Declares the namespaces declared in rFrom only when needed. Moreover, the namespaces are declared in such a way as to avoid namespace prefix clashes.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	Where to declare the namespaces that are declared in rFrom. eNSPOS_AS_COPY can only be used in conjunction with eNSDECLARE_FORCE, while the other positions are possible with eNSDECLARE_OPTIMIZE.

Returns

A pointer to the new element. NULL otherwise.

Description

Creates a child or sibling element that is a copy of another element and attaches it at the requested place.

This method cannot be called on a root element to create a sibling. Only child elements can be created from a root element.

See Also

IXmlDocument::CreateRootElement (see page 875), IXmlDocument::GetRootElement (see page 877), IXmlDocument::DeleteRootElement (see page 876).

2.19.1.1.2.4.2 - CXmLElement::CreateElement Method

Creates a child or sibling element that has its namespace already declared or that is assigned to no namespace.

C++

```
CXmLElement* CreateElement(IN const char* pszNamespace, IN const char* pszElementName, IN const char* pszValue, IN ECreatePosition ePos);
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace of the new element. It can be NULL. When non-NULL, the namespace MUST have been declared in the new element's parents. Otherwise this method will fail.
IN const char* pszElementName	The name of the new element. It must not be NULL.
IN const char* pszValue	The value of the new element.

IN ECreatePosition ePos	The position where the element must be inserted.
	<ul style="list-style-type: none"> • ePOS_FIRST_SIBLING: The new element is created as the first child of the parent element. • ePOS_PREVIOUS_SIBLING: The new element is created and inserted as the previous sibling of this element. • ePOS_NEXT_SIBLING: The new element is created and inserted as the next sibling of this element. • ePOS_LAST_SIBLING: The new element is created and inserted as the last child of the parent element. • ePOS_FIRST_CHILD: The new element is created and inserted as the first child of this element. • ePOS_LAST_CHILD: The new element is created and inserted as the last child of this element.

Returns

A pointer to the new element.

NULL if the insertion failed (for example, if pszElementName was invalid).

Description

Creates an element that is attached at the requested place.

This method cannot be called on a root element to create a sibling. Only child elements can be created from a root element.

See Also

IXmlDocument::CreateRootElement (see page 875), IXmlDocument::GetRootElement (see page 877), IXmlDocument::DeleteRootElement (see page 876).

2.19.1.1.2.4.3 - CXmIElement::CreateElement Method

Creates a child or sibling element and allows to declare the namespace assigned to this new element.

C++

```
CXmlElement* CreateElement(IN const char* pszNamespaceUri, IN const char* pszNamespacePrefix, IN const char* pszElementName, IN const char* pszValue, IN ECreatePosition ePos, IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const char* pszNamespaceUri	The namespace URI of the element to create. The namespace may be declared at this element level depending on the eBehavior parameter. It MUST NOT be NULL.
IN const char* pszNamespacePrefix	The prefix for pszNamespaceUri. It can be NULL if pszNamespaceUri is the default namespace. This parameter is ignored if the eBehavior parameter is set to eOPTIMIZE_NS and pszNamespaceUri is already declared in one of the parents.
IN const char* pszElementName	The name of the new element. It must not be NULL.
IN const char* pszValue	The value of the new element.

IN ECreatePosition ePos	The position where the element must be inserted.
	<ul style="list-style-type: none"> • ePOS_FIRST_SIBLING: The new element is created as the first child of the parent element. • ePOS_PREVIOUS_SIBLING: The new element is created and inserted as the previous sibling of this element. • ePOS_NEXT_SIBLING: The new element is created and inserted as the next sibling of this element. • ePOS_LAST_SIBLING: The new element is created and inserted as the last child of the parent element. • ePOS_FIRST_CHILD: The new element is created and inserted as the first child of this element. • ePOS_LAST_CHILD: The new element is created and inserted as the last child of this element.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	The behavior to have to declare the namespace.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	The position specifying where to declare missing namespaces.

Returns

A pointer to the new element.

NULL if the insertion failed (for example, if pszElementName was invalid).

Description

Creates an element and attaches it at the requested place in relation with this element.

When calling this method, the specified namespace is declared in the created element if is not already declared in a parent element and usable in the created element.

This method cannot be called on a root element to create a sibling. Only child elements can be created from a root element.

See Also

IXmlDocument::CreateRootElement (see page 875), IXmlDocument::GetRootElement (see page 877), IXmlDocument::DeleteRootElement (see page 876), ENsDeclarationBehavior (see page 866), ENsDeclarationPosition (see page 867)

2.19.1.1.2.5 - CXmLElement::DeclareNamespace Method

Declares a namespace in this element.

C++

```
mxt_result DeclareNamespace(IN const char* pszNamespaceUri, IN const char* pszNamespacePrefix, IN
ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition =
eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const char* pszNamespaceUri	The namespace URI to declare in this element. It MUST NOT be NULL.
IN const char* pszNamespacePrefix	The prefix of namespace to declare in this element. It can be NULL if this is the default namespace.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	The behavior when declaring namespaces.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	enum specifying where to declare namespaces.

Returns

- resFE_INVALID_ARGUMENT: pszNamespaceUri is NULL or the namespace prefix was already defined in this element.
- resFE_FAIL: failed to allocate a new SNamespace (see page 842) structure.
- resS_OK: the namespace was successfully declared in this element.

Description

This method declares a namespace in the element.

This namespace declaration applies to this element and its children.

See Also

[ENsDeclarationBehavior](#) (see page 866), [ENsDeclarationPosition](#) (see page 867)

2.19.1.1.2.6 - CXmLElement::Delete Method

Deletes this element.

C++

```
mxt_result Delete();
```

Returns

resS_OK if the element has been deleted, or resFE_ACCESS_DENIED if it is the root element.

Description

This method releases the XML (see page 839) element and all internal resources it holds. If the current element is the document's root element, resFE_ACCESS_DENIED is returned and [IXmlDocument::DeleteRootElement](#) (see page 876) must be called instead.

The caller's pointer becomes invalid when this method returns resS_OK. The caller should set its pointer to NULL.

See Also

[IXmlDocument::DeleteRootElement](#) (see page 876)

2.19.1.1.2.7 - DeleteAttribute**2.19.1.1.2.7.1 - CXmLElement::DeleteAttribute Method**

Deletes the attribute that has the specified name.

C++

```
mxt_result DeleteAttribute(IN const char* pszAttrNamespace, IN const char* pszAttrName);
```

Parameters

Parameters	Description
IN const char* pszAttrNamespace	The namespace of the attribute to delete.
IN const char* pszAttrName	The name of the attribute to delete. It must not be NULL.

Returns

resS_OK if the attribute has been deleted, an error otherwise.

Description

This method releases all the resources taken by the attribute described by the given parameters.

2.19.1.1.2.7.2 - CXmLElement::DeleteAttribute Method

Deletes an attribute at the specified index.

C++

```
mxt_result DeleteAttribute(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the attribute to delete.

Returns

resS_OK if the attribute has been deleted, an error otherwise.

Description

This method releases all the resources taken by the attribute at the given index.

2.19.1.1.2.8 - CXmIElement::DeleteDeclaredNamespace Method

Deletes a declared namespace.

C++

```
mxt_result DeleteDeclaredNamespace(IN const char* pszPrefix);
```

Parameters

Parameters	Description
IN const char* pszPrefix	The prefix of namespace to delete.

Returns

- resS_OK upon success.
- resFE_INVALID_ARGUMENT: pszPrefix does not exist.
- resFE_FAIL: Current element or one of its children uses the namespace.

Description

Deletes a declared namespace. This method will fail if this element or one of its children uses the namespace.

2.19.1.1.2.9 - FindChildElement

2.19.1.1.2.9.1 - CXmIElement::FindChildElement Method

Finds a child element according to its name and index.

C++

```
CXmIElement* FindChildElement(IN const char* pszNamespace, IN const char* pszElementName, IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace to match.
IN const char* pszElementName	The name to match. It must not be NULL.
IN unsigned int uIndex	The number of times the namespace and name must match before returning the element. If 0, the first element that matches is returned.

Returns

The pointer to the child element found with the given parameters. It can be NULL if no element was found.

Description

This method tries to match a number of times (uIndex + 1) the given namespace and name pair. It then returns an element pointer if there is any. So, if uIndex is 0, the first element matched is returned, if 1, the second, and so on.

2.19.1.1.2.9.2 - CXmIElement::FindChildElement Method

Finds a child element according to its name and index.

C++

```
const CXmIElement* FindChildElement(IN const char* pszNamespace, IN const char* pszElementName, IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace to match.
IN const char* pszElementName	The name to match. It must not be NULL.
IN unsigned int uIndex	The number of times the namespace and name must match before returning the element. If 0, the first element that matches is returned.

Returns

The pointer to the child element found with the given parameters. It can be NULL if no element was found.

Description

This method tries to match a number of times (uIndex + 1) the given namespace and name pair. It then returns an element pointer if there is any. If uIndex is 0, the first element matched is returned, if 1 the second, and so on.

2.19.1.1.2.10 - GetAttribute**2.19.1.1.2.10.1 - CXmLElement::GetAttribute Method**

Gets an attribute that has the specified name.

C++

```
mxt_result GetAttribute(IN const char* pszAttrNamespace, IN const char* pszAttrName, OUT const char*& rpszAttrValue) const;
```

Parameters

Parameters	Description
IN const char* pszAttrNamespace	The namespace URI of the attribute to look for.
IN const char* pszAttrName	The name of the attribute to look for. It must not be NULL.
OUT const char*& rpszAttrValue	The value of the attribute.

Returns

resS_OK if the attribute has been found, an error otherwise.

Description

This method gets the value of the attribute described by the given namespace and name.

2.19.1.1.2.10.2 - CXmLElement::GetAttribute Method

Gets an attribute at a specific index.

C++

```
mxt_result GetAttribute(IN unsigned int uIndex, OUT const char*& rpszAttrNamespace, OUT const char*& rpszAttrName, OUT const char*& rpszAttrValue) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the attribute to look for.
OUT const char*& rpszAttrNamespace	The namespace of the attribute.
OUT const char*& rpszAttrName	The name of the attribute.
OUT const char*& rpszAttrValue	The value of the attribute.

Returns

resS_OK if the attribute has been found, an error otherwise.

Description

This method gets the namespace, name and value of the attribute at the given index. Index 0 denotes the first attribute.

2.19.1.1.2.11 - GetChildElement**2.19.1.1.2.11.1 - CXmLElement::GetChildElement Method**

Gets a child element at the specified index.

C++

```
CXmLElement* GetChildElement(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the child element.

Returns

The pointer to the child element at the given index. It can be NULL if no element is at the given index.

Description

This method gets the child element located at the given index.

2.19.1.1.2.11.2 - CXmLElement::GetChildElement Method

Gets a child element at the specified index.

C++

```
const CXmLElement* GetChildElement(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the child element.

Returns

The pointer to the child element at the given index. It can be NULL if no element is at the given index.

Description

This method gets the child element located at the given index.

2.19.1.1.2.12 - CXmLElement::GetDeclaredNamespaces Method

Gets the namespaces declared at this level.

C++

```
SNamespace* GetDeclaredNamespaces();
```

Returns

The list of namespace declared at this level.

Description

Gets the namespaces declared at this level. To iterate through the list, use SNamespace::m_pstNextNamespace (see page 842).

2.19.1.1.2.13 - GetFirstSibling**2.19.1.1.2.13.1 - CXmLElement::GetFirstSibling Method**

Gets the first sibling element from this element.

C++

```
CXmLElement* GetFirstSibling();
```

Returns

The pointer to the first sibling, this element can be returned if it is the first sibling.

Description

This method returns the pointer to the first sibling element from this element.

A sibling element is one that shares the same level as the current element.

2.19.1.1.2.13.2 - CXmLElement::GetFirstSibling Method

Gets the first sibling element from this element.

C++

```
const CXmLElement* GetFirstSibling() const;
```

Returns

The pointer to the first sibling, this element can be returned if it is the first sibling.

Description

This method returns the pointer to the first sibling element from this element.

A sibling element is one that shares the same level as the current element.

2.19.1.1.2.14 - CXmLElement::GetLastSibling

2.19.1.1.2.14.1 - CXmLElement::GetLastSibling Method

Gets the last sibling element from this element.

C++

```
CXmLElement* GetLastSibling();
```

Returns

The pointer to the last sibling, this element can be returned if it is the last sibling.

Description

This method returns the pointer to the last sibling element from this element.

A sibling element is one that shares the same level as the current element.

2.19.1.1.2.14.2 - CXmLElement::GetLastSibling Method

Gets the last sibling element from this element.

C++

```
const CXmLElement* GetLastSibling() const;
```

Returns

The pointer to the last sibling, this element can be returned if it is the last sibling.

Description

This method returns the pointer to the last sibling element from this element.

A sibling element is one that shares the same level as the current element.

2.19.1.1.2.15 - CXmLElement::GetName Method

Gets the name of this element.

C++

```
const char* GetName() const;
```

Returns

The pointer to the element's name.

Description

This method returns the element's name.

2.19.1.1.2.16 - CXmLElement::GetNamespace Method

Gets the namespace for this element.

C++

```
const char* GetNamespace() const;
```

Returns

This element's namespace as a C-style string.

NULL if the namespace was not set yet.

Description

This method returns the element's namespace.

2.19.1.1.2.17 - CXMLElement::GetNamespaceByPrefix Method

Gets the namespace for this element from the namespace prefix.

C++

```
const SNamespace* GetNamespaceByPrefix(IN const char* const pszNamespacePrefix) const;
```

Parameters

Parameters	Description
IN const char* const pszNamespacePrefix	The prefix of namespace to get.

Returns

- The namespace having pszNamespacePrefix as prefix.
- NULL if pszNamespacePrefix does not represent any namespace declared at the element's level.

Description

This method returns the namespace declared at this element level and with the pszNamespacePrefix prefix.

Comparison is case-sensitive.

The returned namespace is a namespace declared in this element or in one of its parents.

2.19.1.1.2.18 - CXMLElement::GetNamespaceByUri Method

Gets the namespace for this element from the namespace URI.

C++

```
const SNamespace* GetNamespaceByUri(IN const char* const pszNamespaceUri) const;
```

Parameters

Parameters	Description
IN const char* const pszNamespaceUri	The URI of namespace to get.

Returns

- The namespace having pszNamespaceUri as URI.
- NULL if pszNamespaceUri does not represent any namespace accessible from this element's level.

Description

This method returns a pointer to a declared namespace that matches pszNamespaceUri and that is in scope for this element. This means that the returned namespace and its prefix are usable in this element.

Comparison is case-sensitive.

The returned namespace is a namespace declared in this element or in one of its parents.

2.19.1.1.2.19 - CXMLElement::GetNamespacePrefix Method

Gets the prefix of the namespace for this element.

C++

```
const char* GetNamespacePrefix() const;
```

Returns

The prefix of this element's namespace as a C-style string.

NULL if the namespace was not set yet or if it is in the default namespace.

Description

This method returns the prefix of this element's namespace.

2.19.1.1.2.20 - CXmLElement::GetNextSibling

2.19.1.1.2.20.1 - CXmLElement::GetNextSibling Method

Gets the next sibling element to this element.

C++

```
CXmLElement* GetNextSibling();
```

Returns

The pointer to the next sibling, or NULL if there is none.

Description

This method returns the pointer to the next sibling element if there is any. A sibling element is one that shares the same level (same parent element) as the current element.

2.19.1.1.2.20.2 - CXmLElement::GetNextSibling Method

Gets the next sibling element to this element.

C++

```
const CXmLElement* GetNextSibling() const;
```

Returns

The pointer to the next sibling, or NULL if there is none.

Description

This method returns the pointer to the next sibling element if there is any. A sibling element is one that shares the same level (same parent element) as the current element.

2.19.1.1.2.21 - CXmLElement::GetNumAttributes Method

Gets the number of attributes held by this element.

C++

```
unsigned int GetNumAttributes() const;
```

Returns

The number of attributes.

Description

This method returns the number of attributes associated with this element.

2.19.1.1.2.22 - CXmLElement::GetNumChildElement Method

Gets the number of child elements held by this element.

C++

```
unsigned int GetNumChildElement() const;
```

Returns

The number of children.

Description

This method returns the number of child elements associated with this element. A child element is contained inside the scope of the current element.

2.19.1.1.2.23 - CXmLElement::GetOpaque Method

Retrieves the opaque identifier.

C++

```
const mxt_opaque GetOpaque();
```

Returns

This CXmLElement (See page 840)'s opaque.

Description

Returns the opaque associated with this CXmLElement (See page 840).

2.19.1.1.2.24 - GetParentElement

2.19.1.1.2.24.1 - CXmLElement::GetParentElement Method

Gets the parent element to this element.

C++

```
CXmLElement* GetParentElement();
```

Returns

The pointer to the parent, or NULL if it is the root element.

Description

This method returns the pointer to the parent element if there is any. A parent element is the one that contains the current element.

2.19.1.1.2.24.2 - CXmLElement::GetParentElement Method

Gets the parent element to this element.

C++

```
const CXmLElement* GetParentElement() const;
```

Returns

The pointer to the parent, or NULL if it is the root element.

Description

This method returns the pointer to the parent element if there is any. A parent element is the one that contains the current element.

2.19.1.1.2.25 - GetPreviousSibling

2.19.1.1.2.25.1 - CXmLElement::GetPreviousSibling Method

Gets the previous sibling element to this element.

C++

```
CXmLElement* GetPreviousSibling();
```

Returns

The pointer to the previous sibling, or NULL if there is none.

Description

This method returns the pointer to the previous sibling element if there is any. A sibling element is one that shares the same level as the current element.

2.19.1.1.2.25.2 - CXmLElement::GetPreviousSibling Method

Gets the previous sibling element to this element.

C++

```
const CXmLElement* GetPreviousSibling() const;
```

Returns

The pointer to the previous sibling, or NULL if there is none.

Description

This method returns the pointer to the previous sibling element if there is any. A sibling element is one that shares the same level as the current element.

2.19.1.1.2.26 - CXmLElement::GetValue Method

Gets the string value associated with this element.

C++

```
const char* GetValue() const;
```

Returns

The pointer to the element's value. It can be NULL if the element has no value.

Description

This method returns the element's value.

2.19.1.1.2.27 - CXmLElement::GetXmlDocument Method

Returns the IXmlDocument (see page 874) value.

C++

```
void GetXmlDocument(OUT IXmlDocument** ppDocument) const;
```

Parameters

Parameters	Description
OUT IXmlDocument** ppDocument	A pointer to the pointer to be filled with the IXmlDocument (see page 874) pointer, or with NULL if an error prevented to get the IXmlDocument (see page 874).

Description

Returns the IXmlDocument (see page 874) value.

2.19.1.1.2.28 - CXmLElement::Serialize Method

Serializes the current element and children into a blob.

C++

```
mxt_result Serialize(IN IXmlGenericWriter& rWriter) const;
```

Parameters

Parameters	Description
IN IXmlGenericWriter& rWriter	The writer to use to serialize the element and its children.

Returns

resS_OK upon success, an error otherwise.

Description

Serializes the current element and children into a blob.

The element is serialized at the current blob insertion point. Previously inserted data is thus kept.

If an element has both a value and child element(s), the value will not be serialized as it is impossible to know which of the children or the value came first.

2.19.1.1.2.29 - CXmLElement::SetAttribute Method

Sets the value of the attribute or creates it if it does not exist.

C++

```
mxt_result SetAttribute(IN const char* pszAttrNamespace, IN const char* pszAttrName, IN const char*
pszNewAttrValue);
```

Parameters

Parameters	Description
IN const char* pszAttrNamespace	The namespace in which pszAttrName is defined. It MUST not be NULL. The namespace MUST be declared.
IN const char* pszAttrName	The name of the attribute. It must not be NULL.
IN const char* pszNewAttrValue	The new value of the attribute. It must not be NULL.

Returns

resS_OK if the update is successful or if the attribute has been added.

Failure if something failed.

Description

Sets the value of the attribute or creates it if it does not exist.

2.19.1.1.2.30 - CXMLElement::SetChildElement Method

Creates or replaces a child element that is inserted following a specified order.

C++

```
CXMLElement* SetChildElement(IN const char* pszNamespace, IN const char* pszNamespacePrefix, IN const char*
pszElementName, IN const char* pszNewValue, IN const CVector<SIIdentificationInfo>* pvecstElementOrder, IN
ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE, IN ENsDeclarationPosition ePosition =
eNSPOS_ROOT_ELEMENT);
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace of the element to set. pszNamespace MUST be present in pvecstElementOrder along with pszElementName UNLESS there is an 'any' element (i.e. {NULL, NULL}) in pvecstElementOrder. In the latter case, the element is searched after the element preceeding the 'any' element and before the element following the 'any' element in pvecstElementOrder. If the element is not existing, it is inserted as the previous sibling of the element following the 'any' element. It MUST NOT be NULL.
IN const char* pszNamespacePrefix	The prefix for pszNamespaceUri. It can be NULL if pszNamespace is the default namespace. This parameter is ignored if the element is already present or if pszNamespace is already declared in one of the parents and it must not be declared.
IN const char* pszElementName	The name of the element to set. pszElementName MUST be present in pvecstElementOrder along with pszNamespace UNLESS there is an 'any' element (i.e. {NULL, NULL}) in pvecstElementOrder. In the latter case, the element is searched after the element preceeding the 'any' element and before the element following the 'any' element in pvecstElementOrder. If the element is not existing, it is inserted as the previous sibling of the element following the 'any' element. It MUST NOT be NULL.
IN const char* pszNewValue	The value to set in the found or created element. It can be NULL.
IN const CVector<SIIdentificationInfo>* pvecstElementOrder	A pointer to a CVector (see page 227) giving the order of the elements. If an element has to be created, it is added at the place its name/namespace are in the CVector (see page 227). To permit undefined names to be present in the children, insert an 'any' SIIdentificationInfo (see page 841) structure (i.e. with a NULL name and a NULL namespace). Note that if many 'any' SIIdentificationInfo (see page 841) are added, only the last one is taken into account. It can be NULL. In this case, an unexisting element is inserted as the last child element.
IN ENsDeclarationBehavior eBehavior = eNSDECLARE_OPTIMIZE	The behavior to have to declare the namespace.
IN ENsDeclarationPosition ePosition = eNSPOS_ROOT_ELEMENT	The position specifying where to declare missing namespaces.

Returns

The element that was set or created.

NULL if an error occurred.

Description

This method searches for the element specified by pszNamespace and pszElementName. If the element is found, it sets its value to psznewValue.

If the element is not found, it is created and inserted at the place where it is in pvecstElementOrder, and its value is set to psznewValue.

This method manages the child as if there is a sequence in which each element can be present only once.

See Also

CreateChildElement (see page 844), CreateElement (see page 848), ENsDeclarationBehavior (see page 866), ENsDeclarationPosition (see page 867).

Example

This shows how the pvecstElementOrder works.

```

CVector<SIIdentificationInfo> vecstElementOrder;

// vecstElementOrder is initialized to contain
// {"urn:ietf:params:xml:ns:pidf:rpid", "note"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "audio"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "video"},
// {"urn:ietf:params:xml:ns:pidf:rpid", "text"}
// (...)

// pXmlElement is a new element containing no child element yet.
CXmlElement* pXmlElement = (...);

CXmlElement* pAudio1 = pXmlElement->SetChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                 "rpid",
                                                 "audio",
                                                 "value1",
                                                 &vecstElementOrder);

// pAudio1 is now set to a newly created element with value of "value1".

// pXmlElement now has the following children elements
// <rpid:audio>value1
//
// Where rpid is the in-scope namespace prefix of
// "urn:ietf:params:xml:ns:pidf:rpid"

CXmlElement* pNote1 = pXmlElement->SetChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                 "rpid",
                                                 "note",
                                                 "note1",
                                                 &vecstElementOrder);

// pNote1 is now set to a newly created element with value of "note1".

// pXmlElement now has the following children elements
// <rpid:note>note1
// <rpid:audio>value1

CXmlElement* pText1 = pXmlElement->SetChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                 "rpid",
                                                 "text",
                                                 "text1",
                                                 &vecstElementOrder);

// pText1 is now set to a newly created element with value of "text1".

// pXmlElement now has the following children elements
// <rpid:note>note1
// <rpid:audio>value1
// <rpid:text>text1

CXmlElement* pAudio2 = pXmlElement->SetChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                 "rpid",
                                                 "audio",
                                                 "value2",
                                                 &vecstElementOrder);

```

```

// pAudio2 is now pointing to pAudio1 (no new object was created). The
// value is now "value2".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <rpid:text>text1

CXmlElement* pVideo1 = pXmlElement->SetChildElement("urn:ietf:params:xml:ns:pidf:rpid",
                                                    "rpid",
                                                    "video",
                                                    "video1",
                                                    &vecstElementOrder);

// pVideo1 is now set to a newly created element with value of "video1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <rpid:video>video1
//  <rpid:text>text1

```

This shows how to use the 'any' element:

```

CVector<SIIdentificationInfo> vecstElementOrder;

// vecstElementOrder is initialized to contain
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "note"},
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "audio"},
//  {NULL,                                     NULL}, // any element
//  {"urn:ietf:params:xml:ns:pidf:rpid",      "video"},
//  {"other:namespace",                      "note"}
(...)

// pXmlElement is an element containing the following children:
//  <rpid:note>note1
//  <rpid:audio>value2
//  <ons:note>endnote1
//
// Where rpid is the in-scope namespace prefix of
// "urn:ietf:params:xml:ns:pidf:rpid"
// and ons is the in-scope namespace prefix of "other:namespace".
CXmlElement* pXmlElement = (...);

CXmlElement* pCustom1 = pXmlElement->SetChildElement("other:namespace",
                                                    "oth",
                                                    "custom",
                                                    "value1",
                                                    &vecstElementOrder);

// pCustom1 is now set to a newly created element with value of "value1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <oth:custom>value1
//  <ons:note>endnote1

CXmlElement* pOther1 = pXmlElement->SetChildElement("other:namespace",
                                                    "oth",
                                                    "other",
                                                    "othvalue1",
                                                    &vecstElementOrder);

// pOther1 is now set to a newly created element with value of "othvalue1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <oth:custom>value1
//  <oth:other>othvalue1
//  <ons:note>endnote1

CXmlElement* pCustom2 = pXmlElement->SetChildElement("other:namespace",
                                                    "oth",
                                                    "custom",
                                                    "value2",
                                                    &vecstElementOrder);

// pCustom2 is now pointing to pCustom1 (no new object was created). The
// value is now "value2".

```

```

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <oth:custom>value2
//  <oth:other>othervalue1
//  <ons:note>endnote1

CXMLElement* pVideo1 = pXmlElement->SetChildElement( "urn:ietf:params:xml:ns:pidf:rpid",
                                                     "rpid",
                                                     "video",
                                                     "video1",
                                                     &vecstElementOrder);

// pVideo1 is now set to a newly created element with value of "video1".

// pXmlElement now has the following children elements
//  <rpid:note>note1
//  <rpid:audio>value2
//  <oth:custom>value2
//  <oth:other>othervalue1
//  <rpid:video>video1
//  <ons:note>endnote1

```

2.19.1.1.2.31 - CXMLElement::SetName Method

Sets the name of this element.

C++

```
void SetName(IN const char* pszName);
```

Parameters

Parameters	Description
IN const char* pszName	The name to set. It MUST not be NULL.

Description

This method sets the name of the element.

2.19.1.1.2.32 - CXMLElement::SetNamespace Method

Sets the namespace for this element.

C++

```
mxt_result SetNamespace(IN const char* pszNamespace);
```

Parameters

Parameters	Description
IN const char* pszNamespace	The namespace to set.

Returns

- resFE_INVALID_ARGUMENT: the namespace is not defined in this element or in one of its parents.
- resS_OK: the namespace was set.

Description

This method sets the namespace of the element. The namespace MUST be first defined in the document before being set on the element, otherwise it won't be set and the namespace's element will be empty.

2.19.1.1.2.33 - CXMLElement::SetOpaque Method

Sets the opaque for this CXMLElement (see page 840).

C++

```
void SetOpaque(mxt_opaque opq);
```

Parameters

Parameters	Description
mxt_opaque opq	The opaque to set.

Description

Sets the opaque associated with this CXmLElement (see page 840).

2.19.1.1.2.34 - CXmLElement::SetValue Method

Sets the string value associated with this element.

C++

```
void SetValue(IN const char* pszValue);
```

Parameters

Parameters	Description
IN const char* pszValue	The value to set. It can be NULL.

Description

This method sets the value of the element.

2.19.1.1.2.35 - UpdateAttribute**2.19.1.1.2.35.1 - CXmLElement::UpdateAttribute Method**

Changes the namespace, name and value associated with an existing attribute.

C++

```
mxt_result UpdateAttribute(IN const char* pszAttrNamespace, IN const char* pszAttrName, IN const char* pszAttrnewValue);
```

Parameters

Parameters	Description
IN const char* pszAttrNamespace	The namespace URI of the attribute to look for.
IN const char* pszAttrName	The name of the attribute to look for.
IN const char* pszAttrnewValue	The new value of the attribute. It must not be NULL.

Returns

resS_OK if the update is successful, an error otherwise.

Description

This method updates the value of the attribute that matches the given namespace and name.

2.19.1.1.2.35.2 - CXmLElement::UpdateAttribute Method

Changes the namespace, name and value associated with an existing attribute.

C++

```
mxt_result UpdateAttribute(IN unsigned int uIndex, IN const char* pszAttrValue);
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the attribute to update.
pszAttrnewValue	The new value of the attribute. It must not be NULL.

Returns

resS_OK if the update is successful, an error otherwise.

Description

This method updates the attribute at the given index with a new value.

2.19.1.1.2.36 - CXmLElement::UpdateDeclaredNamespace Method

Updates the URI of a declared namespace.

C++

```
mxt_result UpdateDeclaredNamespace(IN const char* pszPrefix, IN const char* pszNewUri);
```

Parameters

Parameters	Description
IN const char* pszPrefix	The prefix of namespace to update.
IN const char* pszNewUri	The new URI to set.

Returns

- resS_OK upon success.
- resFE_INVALID_ARGUMENT when pszNewUri is invalid or pszPrefix does not exist.

Description

Updates the URI of a declared namespace. Updating a declared namespace has an impact on all children of this element.

2.19.1.1.3 - Enumerations**2.19.1.1.3.1 - CXMLElement::ECreatePosition Enumeration**

```
enum ECreatePosition {
    ePOS_FIRST_SIBLING,
    ePOS_PREVIOUS_SIBLING,
    ePOS_NEXT_SIBLING,
    ePOS_LAST_SIBLING,
    ePOS_FIRST_CHILD,
    ePOS_LAST_CHILD
};
```

Description

This enumeration is used to specify where a child element is inserted with CreateElement (see page 848).

Members

Members	Description
ePOS_FIRST_SIBLING	Inserts the element as the first sibling element.
ePOS_PREVIOUS_SIBLING	Inserts the element as the previous element (i.e. previous sibling) of the current element.
ePOS_NEXT_SIBLING	Inserts the element as the next element (i.e. next sibling) of the current element.
ePOS_LAST_SIBLING	Inserts the element as the last sibling element.
ePOS_FIRST_CHILD	Inserts the element as the first child element of this element.
ePOS_LAST_CHILD	Inserts the element as the last child element of this element.

2.19.1.1.3.2 - CXMLElement::ENsDeclarationBehavior Enumeration

```
enum ENsDeclarationBehavior {
    eNSDECLARE_NONE,
    eNSDECLARE_FORCE,
    eNSDECLARE_OPTIMIZE,
    eNSDECLARE_OPTIMIZE_FAIL_ON_CONFLICT
};
```

Description

This enumeration defines how and when namespaces are declared.

In the following section, the **used element** is the element upon which is performed an operation that has the effect of potentially creating a namespace. This is the element upon which a method like `DeclareNamespace` (see page 851)(), `CreateElement` (see page 848) or any other method that takes an `ENsDeclarationBehavior` is called. This is not necessarily the same element where the namespace will be declared, as this depends on the `ENsDeclarationPosition` (see page 867) parameter. This later element is called the **specified element**.

Members

Members	Description
eNSDECLARE_NONE	Declares no namespace and fails if the namespace was not previously declared, or if the namespace is not accessible from the <i>*used element*</i> .

eNSDECLARE_FORCE	Forces the declaration of the namespace with the specified prefix. It can only be used when the namespace is created with eNSPOS_THIS_ELEMENT or eNSPOS_NEW_ELEMENT. This method fails if declaring the namespace on eNSPOS_THIS_ELEMENT and the prefix is already declared on this element. Users must be careful when using this value to force the declaration of a namespace, as it is possible that the declared prefix "hides" a previously declared namespace on which a child element depends.
eNSDECLARE_OPTIMIZE	Declares the namespace only if it is not accessible from the *used element*. <ul style="list-style-type: none"> If the namespace is declared but not accessible from the *used element*, the namespace is re-declared in the *specified element*. If needed, the prefix is updated to ensure the namespace is in scope for the *used element* and to ensure that the prefix does not override any existing prefix. If the namespace is not declared, it is declared in the *specified element* and, if needed, the prefix is updated to ensure that it does not override an existing prefix.
eNSDECLARE_OPTIMIZE_FAIL_ON_CONFLICT	Declares the namespace only if it is not accessible from the *used element*. Furthermore, if the namespace has to be declared, it will use the specified prefix. If the specified prefix is not accessible from the *used element* because there is a namespace declaration between the *specified element* and the *used element* that overrides the prefix, the namespace declaration fails.

2.19.1.1.3.3 - CXmLElement::ENsDeclarationPosition Enumeration

```
enum ENsDeclarationPosition {
    eNSPOS_NEW_ELEMENT,
    eNSPOS_THIS_ELEMENT,
    eNSPOS_PARENT_ELEMENT,
    eNSPOS_ROOT_ELEMENT,
    eNSPOS_AS_COPY
};
```

Description

This enumeration describes where a namespace must be created.

In the following section, the *used element* is the element upon which is performed an operation that has the effect of potentially creating a namespace. This is the element upon which a method like `DeclareNamespace` (see page 851)(), `CreateElement` (see page 848) or any other method that takes an ENsDeclarationBehavior (see page 866) is called. This is not necessarily the same element where the namespace will be declared, as this depends on the ENsDeclarationPosition parameter. This later element is called the *specified element*.

Members

Members	Description
eNSPOS_NEW_ELEMENT	The *specified element* where the namespace is created is the new element being created. This value can only be used when creating an element and cannot be used when calling <code>DeclareNamespace</code> (see page 851) or <code>Copy</code> (see page 843).
eNSPOS_THIS_ELEMENT	The *specified element* is the same element as the *used element*.
eNSPOS_PARENT_ELEMENT	The *specified element* is the parent element of the *used element*.
eNSPOS_ROOT_ELEMENT	The *specified element* is the root element.
eNSPOS_AS_COPY	This value can only be used when copying an element into another, or when creating a new element from another source element. The *specified element* is updated dynamically when performing the copy to ensure that namespaces are copied from the source elements to the target elements as they are declared in the source. Note that if an element or attribute from the source uses a namespace that is declared outside of the scope of the copy, then its namespace is declared in the first copied element.

2.19.1.2 - CXmLElement Class

Implements a XML (see page 839) parser.

Class Hierarchy

```
CXmlParser
```

C++

```
class CXmLElement;
```

Description

Class implementing an XML (see page 839) parser. Parses an XML (see page 839) buffer by using the Parse (see page 868) method.

Constructors

Constructor	Description
 CXmlParser (see page 868)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
  ~CXmlParser (see page 868)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 Parse (see page 868)	Parsing method
 SkipElementContent (see page 869)	Parser skip element control.

Legend

	Method
---	--------

2.19.1.2.1 - Constructors**2.19.1.2.1.1 - CXmlParser::CXmlParser Constructor**

Constructor.

C++

```
CXmlParser();
```

Description

Constructor

Constructor

2.19.1.2.2 - Destructors**2.19.1.2.2.1 - CXmlParser::~CXmlParser Destructor**

Destructor.

C++

```
virtual ~CXmlParser();
```

Description

Destructor

2.19.1.2.3 - Methods**2.19.1.2.3.1 - CXmlParser::Parse Method**

Parsing method

C++

```
mxt_result Parse(IN IXmlParserHandler* pXmlParserHandler, IN const uint8_t* puXmlBuffer, IN const uint32_t uBufferSize);
```

Parameters

Parameters	Description
IN IXmlParserHandler* pXmlParserHandler	XML (see page 839) parser manager.
IN const uint8_t* puXmlBuffer	Buffer pointer to the MIB file to parse.
IN const uint32_t uBufferSize	Size of the MIB file to parse.

Returns

- resS_OK
- resFE_INVALID_ARGUMENT
- resFE_FAIL

Description

This function parses a XML (see page 839) file buffer using a XML (see page 839) parser and returns the result to the XML (see page 839) Parser Manager passed as an argument.

2.19.1.2.3.2 - CXmlParser::SkipElementContent Method

Parser skip element control.

C++

```
void SkipElementContent();
```

Description

Parser skip element control.

2.19.1.3 - CXmlWriter Class

Implements methods for writing an XML (see page 839) document.

Class Hierarchy

CXmlWriter

C++

```
class CXmlWriter;
```

Description

Class implementing methods to write an XML (see page 839) document. A document can be written tag by tag or have a large buffer containing the complete document output at once.

Constructors

Constructor	Description
• CXmlWriter (see page 870)	Constructor.

Legend

	Method
---	--------

Destructors

Destructor	Description
• ~CXmlWriter (see page 870)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
EmptyElement (see page 871)	Writes an empty element tag.
EndDocument (see page 871)	Ends an XML (see page 839) Document.
EndElement (see page 871)	Writes an end element tag.
EndElementAttribute (see page 871)	Writes the end of an element attribute and the content value of the element.
EndEmptyElementAttribute (see page 872)	Writes the end of an empty element attribute.
Initialize (see page 872)	Initialize method
StartDocument (see page 872)	Starts an XML (see page 839) Document
StartElement (see page 873)	Writes a start element tag.
StartElementAttribute (see page 873)	Writes the start of an element attribute.
WriteAttributes (see page 873)	Writes an XML (see page 839) attribute.
WriteElement (see page 874)	Writes an XML (see page 839) element.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
EEolMode (see page 874)	Enumeration of the end of line modes.

2.19.1.3.1 - Constructors

2.19.1.3.1.1 - CXmlWriter

2.19.1.3.1.1.1 - CXmlWriter::CXmlWriter Constructor

Constructor.

C++

```
CXmlWriter();
```

Description

Constructor

Constructor

2.19.1.3.1.1.2 - CXmlWriter::CXmlWriter Constructor

Constructor.

C++

```
CXmlWriter(IN IXmlWriterOutputHandler* pXmlWriterOutputHandler, IN EEolMode eEolMode = eLF);
```

Parameters

Parameters	Description
IN IXmlWriterOutputHandler* pXmlWriterOutputHandler	Pointer to an XML (see page 839) output handler.

Description

Creates a XML (see page 839) writer that uses the specified output handler.

Constructor. Creates the writer to write to the specified handler.

2.19.1.3.2 - Destructors

2.19.1.3.2.1 - CXmlWriter::~CXmlWriter Destructor

Destructor.

C++

```
virtual ~CXmlWriter();
```

Description

Destructor

2.19.1.3.3 - Methods**2.19.1.3.3.1 - CXmlWriter::EmptyElement Method**

Writes an empty element tag.

C++

```
bool EmptyElement(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the element.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the empty element is successfully output, false otherwise.

Description

Writes an empty XML (see page 839) element. This is output in the form <pszName/>.

2.19.1.3.3.2 - CXmlWriter::EndDocument Method

Ends an XML (see page 839) Document.

C++

```
bool EndDocument();
```

Returns

True.

Description

Ends the XML (see page 839) document. It asserts if there are any open XML (see page 839) tags.

2.19.1.3.3.3 - CXmlWriter::EndElement Method

Writes an end element tag.

C++

```
bool EndElement(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the element.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the end of an element is successfully output, false otherwise.

Description

Writes the end of an XML (see page 839) element. This is output in the form </pszName>.

2.19.1.3.3.4 - CXmlWriter::EndElementAttribute Method

Writes the end of an element attribute and the content value of the element.

C++

```
bool EndElementAttribute(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT, IN const char* pszValue = NULL);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the attribute.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the start of an element attribute is successfully output, false otherwise.

Description

Writes the end of an XML (see page 839) element attribute and its value if one is specified. This is output in the form >.

2.19.1.3.3.5 - CXmlWriter::EndEmptyElementAttribute Method

Writes the end of an empty element attribute.

C++

```
bool EndEmptyElementAttribute(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the attribute.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the empty end of an element attribute is successfully output, false otherwise.

Description

Writes the end of an empty XML (see page 839) element attribute. This is output in the form />.

2.19.1.3.3.6 - CXmlWriter::Initialize Method

Initialize method

C++

```
void Initialize(IN IXmlWriterOutputHandler* pXmlWriterOutputHandler, IN EEolMode eEolMode = eLF);
```

Parameters

Parameters	Description
IN IXmlWriterOutputHandler* pXmlWriterOutputHandler	Pointer to an XML (see page 839) output handler. It MUST NOT be NULL.

Description

Initializes the CXmlWriter (see page 869) to output to the specified output handler.

2.19.1.3.3.7 - CXmlWriter::StartDocument Method

Starts an XML (see page 839) Document

C++

```
bool StartDocument(IN const char* pszVersion = "1.0", IN const char* pszEncoding = "US-ASCII", IN bool bStandAlone = true);
```

Parameters

Parameters	Description
IN const char* pszVersion = "1.0"	Pointer containing the XML (see page 839) version.
IN const char* pszEncoding = "US-ASCII"	Pointer containing the encoding type.
IN bool bStandAlone = true	bool standalone flag.

Returns

True if the Start of the document has been successfully output, false otherwise.

Description

Writes the XML (see page 839) header containing the XML (see page 839) version, encoding type, and the standalone flag.

2.19.1.3.3.8 - CXmlWriter::StartElement Method

Writes a start element tag.

C++

```
bool StartElement(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the element.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the start of an element was successfully output, false otherwise.

Description

Writes the start of an XML (see page 839) element. This is output in the form <pszName>.

2.19.1.3.3.9 - CXmlWriter::StartElementAttribute Method

Writes the start of an element attribute.

C++

```
bool StartElementAttribute(IN const char* pszName, IN EIndentingMode eIndentingMode = eINDENT);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the attribute.
IN EIndentingMode eIndentingMode = eINDENT	Mode of indentation.

Returns

True if the start of an element attribute is successfully output, false otherwise.

Description

Writes the start of an XML (see page 839) element attribute. This is output in the form <pszName>.

2.19.1.3.3.10 - WriteAttributes**2.19.1.3.3.10.1 - CXmlWriter::WriteAttributes Method**

Writes an XML (see page 839) attribute.

C++

```
bool WriteAttributes(IN const char* pszName, IN const char* pszValue);
bool WriteAttributes(IN const char* pszName, IN int nValue);
bool WriteAttributes(IN const char* pszName, IN unsigned int uValue);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the element attribute.
IN const char* pszValue	Also applies to nValue or uValue, the value associated with this element.

Returns

True if the element attribute is successfully output, false otherwise.

Description

Writes an element attribute tag to the output handler. This is output in the form pszName="value".

2.19.1.3.3.11 - WriteElement

2.19.1.3.3.11.1 - CXmIWriter::WriteElement Method

Writes an XML (see page 839) element.

C++

```
bool WriteElement(IN const char* pszName, IN const char* pszValue);
bool WriteElement(IN const char* pszName, IN int nValue);
bool WriteElement(IN const char* pszName, IN unsigned int uValue);
```

Parameters

Parameters	Description
IN const char* pszName	Pointer to the name of the element.
IN const char* pszValue	Also applies to nValue or uValue, the value associated with this element.

Returns

True if the element is successfully output, false otherwise.

Description

Writes an element tag to the output handler. This is output in the form <pszName>value<pszName/>.

2.19.1.3.4 - Enumerations

2.19.1.3.4.1 - CXmIWriter::EEolMode Enumeration

Enumeration of the end of line modes.

C++

```
enum EEolMode {
    eLF,
    eCR,
    eCRLF,
    eNONE
};
```

Members

Members	Description
eLF	eLF.
eCR	eCR.
eCRLF	eCRLF.
eNONE	eNONE.

2.19.1.4 - IXmlDocument Class

Class Hierarchy



C++

```
class IXmlDocument : public IEComUnknown;
```

Description

The IXmlDocument interface represents an XML (see page 839) document in the Framework's XML (see page 839) Document Object Model (DOM). An XML (see page 839) document is represented as a tree of CXmIElements.

This interface can only handle standalone UTF-8 encoded XML (see page 839) documents.

The document can either be created through parsing of an XML (see page 839) document or by setting values directly into the document.

The user of this interface only has to call the Parse (see page 878) method when the IXmlDocument is used for parsing and accessing an existing document. The root element is accessible through GetRootElement (see page 877). The child elements of the root element are accessible through the various CXMLElement (see page 840) API offered by the root element.

The user of this interface must take the following steps when creating a new XML (see page 839) document:

- Create the XML (see page 839) document using CreateEComInstance (see page 420).
- Create the root element using CreateRootElement (see page 875)
- Use the root element API to add child elements.
- Use the root or children element API to add more elements to the document.
- Use the Serialize (see page 878) method to serialize the document into a buffer.

An IXmlDocument can only handle a single XML (see page 839) document at a time. This document is either created manually or parsed. It cannot be used to parse many different documents without being reset.

Location

Xml/IXmlDocument.h

See Also

CXMLElement (see page 840)

Methods

Method	Description
•   CreateRootElement (see page 875)	Creates the root element.
•   DeleteRootElement (see page 876)	Deletes the root element.
•   GetDictionary (see page 876)	Returns the dictionary for this IXmlDocument.
•   GetDocumentManager (see page 876)	Gets the configured document manager.
•   GetOpaque (see page 877)	Retrieves the opaque identifier.
•   GetPatchManager (see page 877)	Gets the configured patch manager.
•   GetRootElement (see page 877)	Provides access to the root element of the XML (see page 839) document.
•   Parse (see page 878)	Parses a buffer containing an XML (see page 839) document.
•   Serialize (see page 878)	Serializes the current representation of the document into a blob.
•   SetAllocator (see page 879)	Sets the memory allocator to use when creating this XML (see page 839) document.
•   SetDictionary (see page 879)	Sets the dictionary for this IXmlDocument.
•   SetDocumentManager (see page 880)	Sets a document manager that can be notified whenever the document changes.
•   SetOpaque (see page 880)	Sets the opaque for this IXmlDocument.
•   SetPatchManager (see page 880)	Sets a manager to which detailed changes to the document are reported.

IEComUnknown Class

IEComUnknown Class	Description
•   AddRef (see page 418)	Increments the reference count on the ECOM (see page 412) implementing this interface.
•   QueryIf (see page 418)	Queries an object for a supported interface.
•   ReleaseRef (see page 420)	Decrements the reference count on the ECOM (see page 412) implementing this interface.

Legend

	Method
	abstract

2.19.1.4.1 - Methods

2.19.1.4.1.1 - IXmlDocument::CreateRootElement Method

Creates the root element.

C++

```
virtual CXMLElement* CreateRootElement(IN const char* pszNamespace, IN const char* pszNamespacePrefix, IN const char* pszElementName) = 0;
```

Parameters

Parameters	Description
IN const char* pszNamespace	Pointer to the root element's namespace, as a NULL terminated string. This namespace is automatically declared in the root element. It can be NULL.
IN const char* pszNamespacePrefix	Pointer to the prefix for pszNamespace as a NULL terminated string. It can be NULL if pszNamespace is the default namespace. This parameter is ignored when pszNamespace is NULL.
IN const char* pszElementName	Pointer to the root element's name, as a NULL terminated string. It cannot be NULL.

Returns

Pointer to the created root element. NULL pointer if a root element already exists or if the element name is NULL.

Description

Creates and configures the root element. This method is only useful when the caller wants to build an XML (see page 839) document. It is not useful when parsing an XML (see page 839) document and must not be called before Parse (see page 878) is called.

See Also

[CXMLElement](#) (see page 840), [DeleteRootElement](#) (see page 876)

2.19.1.4.1.2 - **IXmlDocument::DeleteRootElement** Method

Deletes the root element.

C++

```
virtual void DeleteRootElement() = 0;
```

Description

Deletes the root element and all its children, if any. After this call, the document is effectively empty.

See Also

[CreateRootElement](#) (see page 875)

2.19.1.4.1.3 - **IXmlDocument::GetDictionary** Method

Returns the dictionary for this IXmlDocument (see page 874).

C++

```
virtual const CVector<const char*>* GetDictionary() = 0;
```

Returns

- The dictionary that was set on this IXmlDocument (see page 874).
- NULL if no IXmlDocument (see page 874) was set on this IXmlDocument (see page 874).

Description

Returns the dictionary for this IXmlDocument (see page 874).

2.19.1.4.1.4 - **IXmlDocument::GetDocumentManager** Method

Gets the configured document manager.

C++

```
virtual IXmlDocumentMgr* GetDocumentManager() = 0;
```

Returns

Pointer to the currently configured document manager.

Description

Gets the configured document manager.

See Also

[SetDocumentManager](#) (see page 880)

2.19.1.4.1.5 - IXmIDocument::GetOpaque Method

Retrieves the opaque identifier.

C++

```
virtual mxt_opaque GetOpaque() = 0;
```

Returns

This IXmIDocument (see page 874)'s opaque.

Description

Returns the opaque associated with this IXmIDocument (see page 874).

2.19.1.4.1.6 - IXmIDocument::GetPatchManager Method

Gets the configured patch manager.

C++

```
virtual IXmlPatchMgr* GetPatchManager() = 0;
```

Returns

Pointer to the currently configured patch manager.

Description

Gets the configured patch manager.

See Also

SetPatchManager (see page 880)

2.19.1.4.1.7 - GetRootElement

2.19.1.4.1.7.1 - IXmIDocument::GetRootElement Method

Provides access to the root element of the XML (see page 839) document.

C++

```
virtual CXmLElement* GetRootElement() = 0;
```

Returns

Pointer to the root element. NULL if no root element has been created yet.

Description

Provides access to the root element of the XML (see page 839) document.

See Also

CXmLElement (see page 840)

2.19.1.4.1.7.2 - IXmIDocument::GetRootElement Method

Provides access to the root element of the XML (see page 839) document.

C++

```
virtual const CXmLElement* GetRootElement() const = 0;
```

Returns

Pointer to the root element. NULL if no root element has been created yet.

Description

Provides access to the root element of the XML (see page 839) document.

See Also

[CXmlElement](#) (see page 840)

2.19.1.4.1.8 - IXmlDocument::Parse Method

Parses a buffer containing an XML (see page 839) document.

C++

```
virtual mxt_result Parse(IN const uint8_t* puXmlBuffer, IN const unsigned int uBufferSize) = 0;
```

Parameters

Parameters	Description
IN const uint8_t* puXmlBuffer	Pointer to the buffer holding the XML (see page 839) document. It cannot be NULL.
IN const unsigned int uBufferSize	The size of the buffer to parse.

Returns

resS_OK upon success, a failure otherwise.

Description

Parses a buffer containing an XML (see page 839) document. The XML (see page 839) namespaces and the root element are automatically configured from the information found in the document. There is no need to create a root element before calling Parse.

Because of the nature of the design, it is not possible to have both child element(s) and value, because there is no way to know in which order they happened. While this does not affect parsing at all, it will affect the parsed document serialization.

So, it would be impossible to know which of this:

```
<test>
    This is a
    <bold>
        TEST
    </bold>
</test>
```

or this would be the correct order.

```
<test>
    <bold>
        TEST
    </bold>
    This is a
</test>
```

Again, this limitation has no impact on the parsing of the document, because both value and children are kept internally. Only the element's serialization will be impacted by this.

Note:

No patch events are reported during parsing and only one IXmlDocumentMgr is reported at the end.

2.19.1.4.1.9 - IXmlDocument::Serialize Method

Serializes the current representation of the document into a blob.

C++

```
virtual mxt_result Serialize(INOUT CBlob& rBlob) = 0;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	The blob into which the XML (see page 839) document is serialized.

Returns

resS_OK upon success, an error otherwise.

Description

Serializes the current representation of the document into a blob.

All defined namespaces are automatically declared as part of the first XML (see page 839) element.

The document is serialized at the current blob insertion point. Previously inserted data is thus kept.

If an element within the document has both a value and child element(s), the value will not be serialized as it is impossible to know which of the children or the value came first.

So, it would be impossible to know which of this:

```
<test>
    This is a
    <bold>
        TEST
    </bold>
</test>
```

or this would be the correct order.

```
<test>
    <bold>
        TEST
    </bold>
    This is a
</test>
```

This is why only the child element(s) will be serialized if both child element(s) and value are set.

2.19.1.4.1.10 - IXmlDocument::SetAllocator Method

Sets the memory allocator to use when creating this XML (see page 839) document.

C++

```
virtual mxt_result SetAllocator(IN IAllocator* pAllocator) = 0;
```

Parameters

Parameters	Description
IN IAllocator* pAllocator	The allocator to use within the document. It can be NULL, in which case the normal MX_NEW (see page 510) and MX_DELETE (see page 509) are used.

Returns

resS_OK upon success, an error if the allocator has already been set or if it cannot be set at this time.

Description

Sets the memory allocator to use within the scope of this XML (see page 839) document. The memory allocator should increase performances for parsed document or otherwise built from known values that shall not change over time. For building a document with no previously known values or otherwise changing values, it is strongly encouraged NOT to set any allocator and use the normal MX_NEW (see page 510) and MX_DELETE (see page 509) operators.

This method must be called first on any new document. The allocator cannot be changed afterwards.

If set, the allocator MUST be valid throughout the life of the document. Thus, the document MUST be released before deleting the allocator it uses.

2.19.1.4.1.11 - IXmlDocument::SetDictionary Method

Sets the dictionary for this IXmlDocument (see page 874).

C++

```
virtual mxt_result SetDictionary(const CVector<const char*>* pvecpszDictionary) = 0;
```

Parameters

Parameters	Description
const CVector<const char*>* pvecpszDictionary	The dictionary to use to find element names, attribute names and namespace URI. It MUST NOT be NULL.

Returns

- resFE_INVALID_ARGUMENT: pvecpszDictionary is NULL.
- resFE_INVALID_STATE: the dictionary was already set.
- resS_OK: the dictionary is set.

Description

Sets the dictionary to use to search element names, attribute names and namespace URI. When one of these is set and its value is found in the dictionary, the C-style string pointer from the dictionary is taken instead of creating a new C-style string on the heap.

2.19.1.4.1.12 - IXmlDocument::SetDocumentManager Method

Sets a document manager that can be notified whenever the document changes.

C++

```
virtual void SetDocumentManager( IN IXmlDocumentMgr* pMgr ) = 0;
```

Parameters

Parameters	Description
IN IXmlDocumentMgr* pMgr	Pointer to the manager to configure. It can be NULL, in which case no event will be reported.

Description

Sets a document manager that can be notified whenever the document changes.

The document manager is only notified once the document has been changed, however the actual changes are not reported. To be notified of the detailed changes, the user should configure a patch manager instead by using SetPatchManager (see page 880).

This API can be called at any time during the life of the IXmlDocument (see page 874) to change the document manager.

See Also

[SetPatchManager](#) (see page 880)

2.19.1.4.1.13 - IXmlDocument::SetOpaque Method

Sets the opaque for this IXmlDocument (see page 874).

C++

```
virtual void SetOpaque( IN mxt_opaque opqData ) = 0;
```

Parameters

Parameters	Description
opq	The opaque to set.

Description

Sets the opaque associated with this IXmlDocument (see page 874).

2.19.1.4.1.14 - IXmlDocument::SetPatchManager Method

Sets a manager to which detailed changes to the document are reported.

C++

```
virtual void SetPatchManager( IN IXmlPatchMgr* pMgr ) = 0;
```

Parameters

Parameters	Description
IN IXmlPatchMgr* pMgr	Pointer to the manager to configure. It can be NULL, in which case no event will be reported.

Description

Sets a manager to which detailed changes to the document are reported.

This API can be called at any time during the life of the IXmlDocument (see page 874) to change the patch manager.

See Also

[SetDocumentManager](#) (see page 880)

2.19.1.5 - IXmlParserHandler Class

Provides the interface for any class that wants to be an XML (see page 839) parser handler.

Class Hierarchy

IXmlParserHandler

C++

```
class IXmlParserHandler;
```

Description

class IXmlParserHandler

This file provides the interface for any class that wants to be an XML (see page 839) Parser handler. In other words, this handler is called with the right method according to the XML (see page 839) being parsed.

All events are synchronous.

Methods

Method	Description
◆ A EvXmlParserHandlerCharacterData (see page 881)	Notifies the parser handler that character data has been detected.
◆ A EvXmlParserHandlerComment (see page 881)	Comment handler. The data is all text inside the comment delimiters.
◆ A EvXmlParserHandlerDefault (see page 882)	Notifies the parser handler that a default element has been detected.
◆ A EvXmlParserHandlerEndElement (see page 882)	Notifies the parser handler that Start or End elements have been detected.
◆ A EvXmlParserHandlerSkippedContent (see page 882)	Notifies the parser handler that a skipped content is available.
◆ A EvXmlParserHandlerStartElement (see page 882)	Notifies the parser handler that Start or End elements have been detected.

Legend

◆	Method
A	abstract

2.19.1.5.1 - Methods

2.19.1.5.1.1 - IXmlParserHandler::EvXmlParserHandlerCharacterData Method

Notifies the parser handler that character data has been detected.

C++

```
virtual void EvXmlParserHandlerCharacterData(IN const char* pcText, IN const unsigned int uTextSize) = 0;
```

Parameters

Parameters	Description
IN const char* pcText	The text string.
IN const unsigned int uTextSize	The size of the text string.

Description

Text handler. The string received is NOT null-terminated. A single block of contiguous text free of mark-ups may still result in a sequence of calls to this handler. If you are searching for a pattern in the text, it may be split across calls to this handler.

2.19.1.5.1.2 - IXmlParserHandler::EvXmlParserHandlerComment Method

Comment handler. The data is all text inside the comment delimiters.

C++

```
virtual void EvXmlParserHandlerComment(IN const char* pszComments) = 0;
```

Parameters

Parameters	Description
IN const char* pszComments	Pointer to a string that holds the comment.

Description

Comment handler. The data is all text inside the comment delimiters.

2.19.1.5.1.3 - IXmIParserHandler::EvXmIParserHandlerDefault Method

Notifies the parser handler that a default element has been detected.

C++

```
virtual void EvXmIParserHandlerDefault(IN const char* pcText, IN const unsigned int uTextSize) = 0;
```

Parameters

Parameters	Description
IN const char* pcText	The text string.
IN const unsigned int uTextSize	The size of the text string.

Description

Default handler. Sets a handler for any character that would not be handled otherwise. This includes both data for which no handlers can be set and data that could be reported but currently has no handler set.

2.19.1.5.1.4 - IXmIParserHandler::EvXmIParserHandlerSkippedContent Method

Notifies the parser handler that a skipped content is available.

C++

```
virtual void EvXmIParserHandlerSkippedContent(IN const char* pcSkippedContent, IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN const char* pcSkippedContent	A pointer to a non NULL terminated skipped content.
IN unsigned int uSize	The size of the skipped content.

Description

Skipped content handler. This handler is called only if CXmIParser::SkipElementContent (see page 869) was formerly called from EvXmIParserHandlerStartElement (see page 882). It is called once for an entire skipped construct content.

A call to this handler is immediately followed by a call to EvXmIParserHandlerEndElement (see page 882) associated with the start element for which the content was to be skipped.

2.19.1.5.1.5 - IXmIParserHandler::EvXmIParserHandlerStartElement Method

Notifies the parser handler that Start or End elements have been detected.

C++

```
virtual void EvXmIParserHandlerStartElement(IN const char* pszElementName, IN CMarshaler* pAttributes, IN const uint32_t uOffset) = 0;
virtual void EvXmIParserHandlerEndElement(IN const char* pszElementName, IN const uint32_t uOffset) = 0;
```

Parameters

Parameters	Description
IN const char* pszElementName	The element's name.
IN CMarshaler* pAttributes	Attributes pairs.
IN const uint32_t uOffset	Offset of the element.

Description

Handler for start and end elements tags. Attributes are marshaled in pairs. The attribute name followed by the attribute value. The offset is the number of bytes between the start element tag and the beginning of the buffer.

2.19.1.6 - IXmIWriterOutputHandler Class

This file provides the interface for any class that wants to be an XmlWriter Manager.

Class Hierarchy

```
IXmIWriterOutputHandler
```

C++

```
class IXmlWriterOutputHandler;
```

Description

```
class IXmlWriterOutputHandler
```

This file provides the interface for any class that wants to be an XmlWriter Manager.

Methods

Method	Description
  EvXmlWriterOutputHandlerWrite (see page 883)	Informs the XML (see page 839) output handler that it can write XML (see page 839) data to the desired destination.

Legend

	Method
	abstract

2.19.1.6.1 - Methods**2.19.1.6.1.1 - IXmlWriterOutputHandler::EvXmlWriterOutputHandlerWrite Method**

Informs the XML (see page 839) output handler that it can write XML (see page 839) data to the desired destination.

C++

```
virtual bool EvXmlWriterOutputHandlerWrite(IN const char* pcBuffer, IN unsigned int uBufferSize) = 0;
```

Parameters

Parameters	Description
IN const char* pcBuffer	A pointer to the buffer containing the XML (see page 839) data to be written.
IN unsigned int uBufferSize	The size of the XML (see page 839) buffer.

Returns

True if the write operation is a success, false otherwise.

Description

Writes up to uBufferLen characters to an output source implemented by the XmlWriterOutputHandler. The uBufferLen contains the number of characters in the string, excluding the terminal NULL.

2.19.2 - Enumerations

This section documents the enumerations of the Sources/Xml folder.

Enumerations

Enumeration	Description
EIndentingMode (see page 883)	Enumeration of the indentation modes.

2.19.2.1 - CXmWriter::EIndentingMode Enumeration

Enumeration of the indentation modes.

C++

```
enum EIndentingMode {
    eINDENT,
    eCOMPACT
};
```

Members

Members	Description
eINDENT	Normal indentation mode. Uses a tab character for indentation.
eCOMPACT	Compact indentation mode.

Index

+

+ 128, 129

+ function 128, 129

<

<< 118, 120, 129

<< function 118, 120, 129

>

>> 118, 121, 129

>> function 118, 121, 129

A

AES crypto algorithm configuration macros 280

AsciiToHex 18

AsciiToHex function 18

Assertion Mechanism Overview 5

B

Base64 crypto algorithm configuration macros 282

Basic 10

C

CAATree 158

CAATree template 158

[] 169

~CAATree 162

= 169

Allocate 162

AllocateSorted 162

CAATree 161

Erase 163

EraseAll 163

EraseAllocated 163

EraseElement 163

EraseSorted 163

FindPtr 164

FindPtrSorted 164

GetAt 164

GetCapacity 165

GetEndIndex 165

GetFirstIndex 165

GetLastIndex 165

GetLockCapacity 166

GetMaxElementIndex 166

GetMinElementIndex 166

GetSize 166

Insert 167

InsertAllocated 167

IsEmpty 167

IsFull 167

LockCapacity 168

ReduceCapacity 168

ReserveCapacity 168

SetComparisonFunction 168

UnlockCapacity 169

CAATree::[] 169

CAATree::~CAATree 162

CAATree::= 169

CAATree::Allocate 162

CAATree::AllocateSorted 162

CAATree::CAATree 161

CAATree::Erase 163

CAATree::EraseAll 163

CAATree::EraseAllocated 163

CAATree::EraseElement 163

CAATree::EraseSorted 163

CAATree::FindPtr 164

CAATree::FindPtrSorted 164

CAATree::GetAt 164

CAATree::GetCapacity 165

CAATree::GetEndIndex 165

CAATree::GetFirstIndex 165

CAATree::GetLastIndex 165

CAATree::GetLockCapacity 166

CAATree::GetMaxElementIndex 166

CAATree::GetMinElementIndex 166

CAATree::GetSize 166

CAATree::Insert 167

CAATree::InsertAllocated 167

CAATree::IsEmpty 167

CAATree::IsFull 167

CAAATree::LockCapacity 168
 CAAATree::ReduceCapacity 168
 CAAATree::ReserveCapacity 168
 CAAATree::SetComparisonFunction 168
 CAAATree::UnlockCapacity 169
 CAes 319
 CAes class 319
 ~CAes 320
 Begin 320, 321
 CAes 320
 End 321, 322
 GetAlgorithm 322
 GetBlockSizeInBits 322
 GetBlockSizeInByte 322
 GetSupportedModes 322
 SetDefaultAction 323
 SetDefaultIV 323
 SetDefaultKey 324
 SetDefaultMode 325
 Update 325
 CAes::~CAes 320
 CAes::Begin 320, 321
 CAes::CAes 320
 CAes::End 321, 322
 CAes::GetAlgorithm 322
 CAes::GetBlockSizeInBits 322
 CAes::GetBlockSizeInByte 322
 CAes::GetSupportedModes 322
 CAes::SetDefaultAction 323
 CAes::SetDefaultIV 323
 CAes::SetDefaultKey 324
 CAes::SetDefaultMode 325
 CAes::Update 325
 CAignedVariableStorage 64
 CAignedVariableStorage template 64
 _Type& 67
 ~CAignedVariableStorage 66
 = 68
 CAignedVariableStorage 66
 Construct 66
 Destruct 67
 Instance 67
 CAignedVariableStorage::_Type& 67
 CAignedVariableStorage::~CAignedVariableStorage 66
 CAignedVariableStorage::= 68
 CAignedVariableStorage::CAignedVariableStorage 66
 CAignedVariableStorage::Construct 66
 CAignedVariableStorage::Destruct 67
 CAignedVariableStorage::Instance 67
 CAliveObj 464
 CAliveObj class 464
 ~CAliveObj 466
 Activate 466
 CAliveObj 466
 IsAlive 467
 IsDead 467
 IsUnborn 467
 CAliveObj::~CAliveObj 466
 CAliveObj::Activate 466
 CAliveObj::CAliveObj 466
 CAliveObj::IsAlive 467
 CAliveObj::IsDead 467
 CAliveObj::IsUnborn 467
 Call stack tracing macros 53
 CAternateName 664
 CAternateName class 664
 != 667
 ~CAternateName 665
 = 668
 == 668
 CAternateName 665
 GetDnsName 665
 GetIpAddress 666
 GetNameCount 666
 GetNameType 666
 GetRfc822Name 667
 GetUniformResourceIdentifier 667
 CAternateName::!= 667
 CAternateName::~CAternateName 665
 CAternateName::= 668
 CAternateName::== 668
 CAternateName::CAternateName 665
 CAternateName::GetDnsName 665
 CAternateName::GetIpAddress 666
 CAternateName::GetNameCount 666
 CAternateName::GetNameType 666

CAAlternateName::GetRfc822Name 667
 CAAlternateName::GetUniformResourceIdentifier 667
 Cap 94
 CAsyncResolver 741
 CAsyncResolver class 741

- Cancel 743
- DisableCacheNonAuthoritativeResponse 744
- EnableCacheNonAuthoritativeResponse 744
- EvQueryCanceledA 744
- EvResponseReceivedA 744
- GetEnumUrisA 745
- GetHostByAddressA 745
- GetHostByNameA 745
- GetHostFile 746
- GetInstance 746
- GetNameServers 746
- GetNamingAuthorityPointersA 746
- GetServicesA 747
- SetNameServers 747

 CAsyncResolver::Cancel 743
 CAsyncResolver::DisableCacheNonAuthoritativeResponse 744
 CAsyncResolver::EnableCacheNonAuthoritativeResponse 744
 CAsyncResolver::EvQueryCanceledA 744
 CAsyncResolver::EvResponseReceivedA 744
 CAsyncResolver::GetEnumUrisA 745
 CAsyncResolver::GetHostByAddressA 745
 CAsyncResolver::GetHostByNameA 745
 CAsyncResolver::GetHostFile 746
 CAsyncResolver::GetInstance 746
 CAsyncResolver::GetNameServers 746
 CAsyncResolver::GetNamingAuthorityPointersA 746
 CAsyncResolver::GetServicesA 747
 CAsyncResolver::SetNameServers 747
 CAsyncSocketFactory 523
 CAsyncSocketFactory class 523

- CreateAsyncSocket 524
- GetSocketList 524
- IsAsyncSocketInList 525
- RegisterConfigurationMgr 525
- RegisterCreationMgr 525
- RemoveSocketFromFactoryList 526
- UnregisterConfigurationMgr 526
- UnregisterCreationMgr 527

 CAsyncSocketFactory::CreateAsyncSocket 524
 CAsyncSocketFactory::GetSocketList 524
 CAsyncSocketFactory::IsAsyncSocketInList 525
 CAsyncSocketFactory::RegisterConfigurationMgr 525
 CAsyncSocketFactory::RegisterCreationMgr 525
 CAsyncSocketFactory::RemoveSocketFromFactoryList 526
 CAsyncSocketFactory::UnregisterConfigurationMgr 526
 CAsyncSocketFactory::UnregisterCreationMgr 527
 CAsyncTlsSocketFactoryCreationMgr 794
 CAsyncTlsSocketFactoryCreationMgr class 794

- ~CAsyncTlsSocketFactoryCreationMgr 796
- CAsyncTlsSocketFactoryCreationMgr 795
- EvCreationRequested 796

 CAsyncTlsSocketFactoryCreationMgr::~CAsyncTlsSocketFactoryCreationMgr 796
 CAsyncTlsSocketFactoryCreationMgr::CAsyncTlsSocketFactoryCreationMgr 795
 CAsyncTlsSocketFactoryCreationMgr::EvCreationRequested 796
 CAtomicOperations 515
 CAtomicOperations template 515

- ~CAtomicOperations 517
- CAtomicOperations 516
- Decrement 517
- Exchange 517
- Increment 517
- Retrieve 518

 CAtomicOperations::~CAtomicOperations 517
 CAtomicOperations::CAtomicOperations 516
 CAtomicOperations::Decrement 517
 CAtomicOperations::Exchange 517
 CAtomicOperations::Increment 517
 CAtomicOperations::Retrieve 518
 CAtomicValue 518
 CAtomicValue template 518

- CAtomicValue 520
- Decrement 520
- Exchange 520
- Increment 521
- Retrieve 521

 CAtomicValue::CAtomicValue 520
 CAtomicValue::Decrement 520
 CAtomicValue::Exchange 520

CAtomicValue::Increment	521	Get	70
CAtomicValue::Retrieve	521	Release	70
CAuthorityKeyIdentifier	668	Reset	71
CAuthorityKeyIdentifier class	668	CAutoPtr::!=	71
!=	673	CAutoPtr::*	71
~CAuthorityKeyIdentifier	670	CAutoPtr::~CAutoPtr	70
=	673	CAutoPtr::<	72
==	673	CAutoPtr::<=	72
CAuthorityKeyIdentifier	669, 670	CAutoPtr::=	72, 73
GetCertificateIssuerCount	670	CAutoPtr::==	73
GetCertificateSerialNumber	671	CAutoPtr::>	73
GetDnsIssuer	671	CAutoPtr::->	74
GetIpAddressIssuer	671	CAutoPtr::>=	74
GetIssuerType	671	CAutoPtr::CAutoPtr	69
GetKeyIdentifier	672	CAutoPtr::Get	70
GetRfc822NameIssuer	672	CAutoPtr::Release	70
GetUniformResourceIdentifierIssuer	672	CAutoPtr::Reset	71
CAuthorityKeyIdentifier::!=	673	CBase64	326
CAuthorityKeyIdentifier::~CAuthorityKeyIdentifier	670	CBase64 class	326
CAuthorityKeyIdentifier::=	673	~CBase64	327
CAuthorityKeyIdentifier::==	673	Begin	327, 328
CAuthorityKeyIdentifier::CAuthorityKeyIdentifier	669, 670	CBase64	327
CAuthorityKeyIdentifier::GetCertificateIssuerCount	670	End	328, 329
CAuthorityKeyIdentifier::GetCertificateSerialNumber	671	GetAlgorithm	329
CAuthorityKeyIdentifier::GetDnsIssuer	671	GetBlockSizeInBits	329
CAuthorityKeyIdentifier::GetIpAddressIssuer	671	GetBlockSizeInByte	329
CAuthorityKeyIdentifier::GetIssuerType	671	GetSupportedModes	329
CAuthorityKeyIdentifier::GetKeyIdentifier	672	GetVariant	330
CAuthorityKeyIdentifier::GetRfc822NameIssuer	672	SetVariant	330
CAuthorityKeyIdentifier::GetUniformResourceIdentifierIssuer	672	Update	330
CAutoPtr	68	CBase64::~CBase64	327
CAutoPtr template	68	CBase64::Begin	327, 328
!=	71	CBase64::CBase64	327
*	71	CBase64::End	328, 329
~CAutoPtr	70	CBase64::EVariant	409
<	72	CBase64::EVariant enumeration	409
<=	72	CBase64::GetAlgorithm	329
=	72, 73	CBase64::GetBlockSizeInBits	329
==	73	CBase64::GetBlockSizeInByte	329
>	73	CBase64::GetSupportedModes	329
->	74	CBase64::GetVariant	330
>=	74	CBase64::SetVariant	330
CAutoPtr	69	CBase64::Update	330

CBasicConstraints 674
 CBasicConstraints class 674
 != 676
 ~CBasicConstraints 675
 = 676
 == 677
 CBasicConstraints 675
 GetPathLengthConstraint 675
 IsACertificateAuthority 676
 CBasicConstraints::!= 676
 CBasicConstraints::~CBasicConstraints 675
 CBasicConstraints::= 676
 CBasicConstraints::== 677
 CBasicConstraints::CBasicConstraints 675
 CBasicConstraints::GetPathLengthConstraint 675
 CBasicConstraints::IsACertificateAuthority 676
 CBinarySemaphore 467
 CBinarySemaphore class 467
 ~CBinarySemaphore 468
 CBinarySemaphore 468
 GetHandle 469
 Signal 469
 Wait 469
 CBinarySemaphore::~CBinarySemaphore 468
 CBinarySemaphore::CBinarySemaphore 468
 CBinarySemaphore::EInitValue 504
 CBinarySemaphore::EInitValue enumeration 504
 CBinarySemaphore::GetHandle 469
 CBinarySemaphore::Signal 469
 CBinarySemaphore::Wait 469
 CBlob 95
 CBlob class 95
 != 111
 [] 111
 ~CBlob 97
 < 112
 <= 112
 = 112
 == 113
 > 113
 >= 113
 Append 98, 99
 AppendBits 99
 CBlob 96, 97
 Erase 100
 EraseAll 100
 GetAt 100
 GetCapacity 101
 GetEndIndex 101
 GetEndIndexPtr 101
 GetFirstIndex 102
 GetFirstIndexPtr 102
 GetLastIndex 102
 GetLockCapacity 103
 GetSize 103
 GetUnreadBits 103
 GetUnreadSize 103
 Insert 104, 105
 IsEmpty 105
 IsFull 105
 LockCapacity 106
 Merge 106
 PadBits 106
 Read 106
 ReadBits 107
 ReadNoCopy 107
 RealignAppendBits 108
 RealignReadBits 108
 ReduceCapacity 108
 ReserveCapacity 109
 Resize 109
 SetReadIndex 110
 SkipBits 110
 Split 110
 Swap 111
 UnlockCapacity 111
 CBlob::!= 111
 CBlob::[] 111
 CBlob::~CBlob 97
 CBlob::< 112
 CBlob::<= 112
 CBlob::= 112
 CBlob::== 113
 CBlob::> 113
 CBlob::>= 113
 CBlob::Append 98, 99

CBlob::AppendBits 99	Reduce 116
CBlob::CBlob 96, 97	Reserve 116
CBlob::Erase 100	CBlockAllocator::!= 117
CBlob::EraseAll 100	CBlockAllocator::~CBlockAllocator 115
CBlob::GetAt 100	CBlockAllocator::== 117
CBlob::GetCapacity 101	CBlockAllocator::CBlockAllocator 115
CBlob::GetEndIndex 101	CBlockAllocator::GetBlockCount 116
CBlob::GetEndIndexPtr 101	CBlockAllocator::Pop 116
CBlob::GetFirstIndex 102	CBlockAllocator::Push 116
CBlob::GetFirstIndexPtr 102	CBlockAllocator::Reduce 116
CBlob::GetLastIndex 102	CBlockAllocator::Reserve 116
CBlob::GetLockCapacity 103	CCertificate 677
CBlob::GetSize 103	CCertificate class 677
CBlob::GetUnreadBits 103	!= 690
CBlob::GetUnreadSize 103	~CCertificate 679
CBlob::Insert 104, 105	= 690
CBlob::IsEmpty 105	== 690
CBlob::IsFull 105	CCertificate 678, 679
CBlob::LockCapacity 106	DisplayCertificate 679
CBlob::Merge 106	GetAuthorityKeyIdentifierExtension 680
CBlob::PadBits 106	GetBasicConstraintsExtension 680
CBlob::Read 106	GetExtendedKeyUsageExtension 680
CBlob::ReadBits 107	GetExtension 681
CBlob::ReadNoCopy 107	GetExtensionByType 681
CBlob::RealignAppendBits 108	GetExtensionCount 681
CBlob::RealignReadBits 108	GetIssuer 682
CBlob::ReduceCapacity 108	GetIssuerAlternateNameExtension 682
CBlob::ReserveCapacity 109	GetKeyUsageExtension 682
CBlob::Resize 109	GetNetscapeCertificateTypeExtension 682
CBlob::SetReadIndex 110	GetNotAfterTime 683
CBlob::SkipBits 110	GetNotBeforeTime 683
CBlob::Split 110	GetPrivateKey 683
CBlob::Swap 111	GetPublicKey 684
CBlob::UnlockCapacity 111	GetSerialNumber 684
CBlockAllocator 114	GetSignature 684
CBlockAllocator class 114	GetSubject 685
!= 117	GetSubjectAlternateNameExtension 685
~CBlockAllocator 115	GetSubjectKeyIdentifierExtension 685
== 117	GetVersion 686
CBlockAllocator 115	IsIssuedBy 686
GetBlockCount 116	IsSelfIssued 686
Pop 116	Restore 687
Push 116	RestoreDer 687

RestorePem 687	CCertificateChain class 691
SetPrivateKey 688	~CCertificateChain 693
Store 688	= 696
StoreDer 688	== 696
StorePem 689	CCertificateChain 692, 693
VerifySignature 689	Clear 693
CCertificate::!= 690	DisplayCertificateChain 693
CCertificate::~CCertificate 679	Extend 693, 694
CCertificate::= 690	GetCertificates 694
CCertificate::== 690	GetEndEntityCertificate 695
CCertificate::CCertificate 678, 679	IsEmpty 695
CCertificate::DisplayCertificate 679	RemoveHighest 695
CCertificate::GetAuthorityKeyIdentifierExtension 680	RemoveLowest 695
CCertificate::GetBasicConstraintsExtension 680	CCertificateChain::~CCertificateChain 693
CCertificate::GetExtendedKeyUsageExtension 680	CCertificateChain::= 696
CCertificate::GetExtension 681	CCertificateChain::== 696
CCertificate::GetExtensionByType 681	CCertificateChain::CCertificateChain 692, 693
CCertificate::GetExtensionCount 681	CCertificateChain::Clear 693
CCertificate::GetIssuer 682	CCertificateChain::DisplayCertificateChain 693
CCertificate::GetIssuerAlternateNameExtension 682	CCertificateChain::Extend 693, 694
CCertificate::GetKeyUsageExtension 682	CCertificateChain::GetCertificates 694
CCertificate::GetNetscapeCertificateTypeExtension 682	CCertificateChain::GetEndEntityCertificate 695
CCertificate::GetNotAfterTime 683	CCertificateChain::IsEmpty 695
CCertificate::GetNotBeforeTime 683	CCertificateChain::RemoveHighest 695
CCertificate::GetPrivateKey 683	CCertificateChain::RemoveLowest 695
CCertificate::GetPublicKey 684	CCertificateChainValidation 696
CCertificate::GetSerialNumber 684	CCertificateChainValidation class 696
CCertificate::GetSignature 684	~CCertificateChainValidation 697
CCertificate::GetSubject 685	CCertificateChainValidation 697
CCertificate::GetSubjectAlternateNameExtension 685	SetTrustedCertificates 698
CCertificate::GetSubjectKeyIdentifierExtension 685	SetUntrustedCertificates 698
CCertificate::GetVersion 686	Validate 698
CCertificate::IsIssuedBy 686	CCertificateChainValidation::~CCertificateChainValidation 697
CCertificate::IsSelfIssued 686	CCertificateChainValidation::CCertificateChainValidation 697
CCertificate::Restore 687	CCertificateChainValidation::SetTrustedCertificates 698
CCertificate::RestoreDer 687	CCertificateChainValidation::SetUntrustedCertificates 698
CCertificate::RestorePem 687	CCertificateChainValidation::Validate 698
CCertificate::SetPrivateKey 688	CCertificateExtension 699
CCertificate::Store 688	CCertificateExtension class 699
CCertificate::StoreDer 688	!= 703
CCertificate::StorePem 689	~CCertificateExtension 700
CCertificate::VerifySignature 689	= 703
CCertificateChain 691	== 704

CCertificateExtension 700
 GetBasicConstraints 700
 GetExtendedKeyUsage 701
 GetIssuerAlternateName 701
 GetKeyUsage 701
 GetNetscapeCertificateType 702
 GetSubjectAlternateName 702
 GetType 702
 IsCritical 703
 CCertificateExtension::!= 703
 CCertificateExtension::~CCertificateExtension 700
 CCertificateExtension::= 703
 CCertificateExtension::== 704
 CCertificateExtension::CCertificateExtension 700
 CCertificateExtension::GetBasicConstraints 700
 CCertificateExtension::GetExtendedKeyUsage 701
 CCertificateExtension::GetIssuerAlternateName 701
 CCertificateExtension::GetKeyUsage 701
 CCertificateExtension::GetNetscapeCertificateType 702
 CCertificateExtension::GetSubjectAlternateName 702
 CCertificateExtension::GetType 702
 CCertificateExtension::IsCritical 703
 CCertificateIssuer 704
 CCertificateIssuer class 704
 != 706
 ~CCertificateIssuer 705
 = 707
 == 707
 CCertificateIssuer 705
 GetName 706
 GetNames 706
 CCertificateIssuer::!= 706
 CCertificateIssuer::~CCertificateIssuer 705
 CCertificateIssuer::= 707
 CCertificateIssuer::== 707
 CCertificateIssuer::CCertificateIssuer 705
 CCertificateIssuer::GetName 706
 CCertificateIssuer::GetNames 706
 CCertificateSubject 707
 CCertificateSubject class 707
 != 710
 ~CCertificateSubject 709
 = 710
 == 710
 CCertificateSubject 708
 GetName 709
 GetNames 709
 CCertificateSubject::!= 710
 CCertificateSubject::~CCertificateSubject 709
 CCertificateSubject::= 710
 CCertificateSubject::== 710
 CCertificateSubject::CCertificateSubject 708
 CCertificateSubject::GetName 709
 CCertificateSubject::GetNames 709
 CCipher 331
 CCipher class 331
 ~CCipher 332
 Begin 333
 CCipher 332
 End 333, 334
 GetAlgorithm 334
 GetBlockSizeInBits 334
 GetBlockSizeInByte 334
 GetSupportedModes 335
 Update 335
 CCipher::~CCipher 332
 CCipher::Begin 333
 CCipher::CCipher 332
 CCipher::EAction 410
 CCipher::EAction enumeration 410
 CCipher::EAlgorithm 410
 CCipher::EAlgorithm enumeration 410
 CCipher::EMode 410
 CCipher::EMode enumeration 410
 CCipher::End 333, 334
 CCipher::GetAlgorithm 334
 CCipher::GetBlockSizeInBits 334
 CCipher::GetBlockSizeInByte 334
 CCipher::GetSupportedModes 335
 CCipher::Update 335
 CCountingSemaphore 470
 CCountingSemaphore class 470
 ~CCountingSemaphore 471
 CCountingSemaphore 471
 GetHandle 471
 Signal 471

Wait 472
 CCountingSemaphore::~CCountingSemaphore 471
 CCountingSemaphore::CCountingSemaphore 471
 CCountingSemaphore::EInitValue 504
 CCountingSemaphore::EInitValue enumeration 504
 CCountingSemaphore::GetHandle 471
 CCountingSemaphore::Signal 471
 CCountingSemaphore::Wait 472
 CCrc 336
 CCrc class 336
 ~CCrc 337
 Begin 337
 CCrc 337
 End 337, 338
 Update 338
 CCrc::~CCrc 337
 CCrc::Begin 337
 CCrc::CCrc 337
 CCrc::ECrcType 410
 CCrc::ECrcType enumeration 410
 CCrc::End 337, 338
 CCrc::Update 338
 CCrypto 339
 CCrypto class 339
 Enter 339
 Exit 339
 Instance 340
 MocanaLog 340
 CCrypto::Enter 339
 CCrypto::Exit 339
 CCrypto::Instance 340
 CCrypto::MocanaLog 340
 CDiffieHellman 340
 CDiffieHellman class 340
 ~CDiffieHellman 344
 = 351
 CDiffieHellman 343, 344
 GeneratePrime 344
 GeneratePublicAndPrivateKeys 345
 GenerateSharedKey 345, 346
 GetGenerator 346
 GetPrime 346, 347
 GetPrivateKey 347, 348
 GetPublicKey 348, 349
 GetSharedKey 349
 ms_auOAKLEY1_PRIME 342
 ms_auOAKLEY14_PRIME 342
 ms_auOAKLEY15_PRIME 342
 ms_auOAKLEY16_PRIME 342
 ms_auOAKLEY17_PRIME 342
 ms_auOAKLEY18_PRIME 342
 ms_auOAKLEY2_PRIME 342
 ms_auOAKLEY5_PRIME 342
 ms_uOAKLEY1_GENERATOR 343
 ms_uOAKLEY14_GENERATOR 343
 ms_uOAKLEY15_GENERATOR 343
 ms_uOAKLEY16_GENERATOR 343
 ms_uOAKLEY17_GENERATOR 343
 ms_uOAKLEY18_GENERATOR 343
 ms_uOAKLEY2_GENERATOR 343
 ms_uOAKLEY5_GENERATOR 343
 SetParameters 350
 CDiffieHellman::~CDiffieHellman 344
 CDiffieHellman::= 351
 CDiffieHellman::CDiffieHellman 343, 344
 CDiffieHellman::GeneratePrime 344
 CDiffieHellman::GeneratePublicAndPrivateKeys 345
 CDiffieHellman::GenerateSharedKey 345, 346
 CDiffieHellman::GetGenerator 346
 CDiffieHellman::GetPrime 346, 347
 CDiffieHellman::GetPrivateKey 347, 348
 CDiffieHellman::GetPublicKey 348, 349
 CDiffieHellman::GetSharedKey 349
 CDiffieHellman::ms_auOAKLEY1_PRIME 342
 CDiffieHellman::ms_auOAKLEY14_PRIME 342
 CDiffieHellman::ms_auOAKLEY15_PRIME 342
 CDiffieHellman::ms_auOAKLEY16_PRIME 342
 CDiffieHellman::ms_auOAKLEY17_PRIME 342
 CDiffieHellman::ms_auOAKLEY18_PRIME 342
 CDiffieHellman::ms_uOAKLEY2_PRIME 342
 CDiffieHellman::ms_uOAKLEY5_PRIME 342
 CDiffieHellman::ms_uOAKLEY1_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY14_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY15_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY16_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY17_GENERATOR 343

CDiffieHellman::ms_uOAKLEY18_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY2_GENERATOR 343
 CDiffieHellman::ms_uOAKLEY5_GENERATOR 343
 CDiffieHellman::SetParameters 350
 CDnsPacket 747
 CDnsPacket class 747
 ~CDnsPacket 748
 = 749
 CDnsPacket 748
 class CHostFile 749
 class CResolverCache 749
 ReleaseExpiredRecords 749
 ReleaseRecord 749
 UpdateRecords 749
 CDnsPacket::~CDnsPacket 748
 CDnsPacket::= 749
 CDnsPacket::CDnsPacket 748
 CDnsPacket::ReleaseExpiredRecords 749
 CDnsPacket::ReleaseRecord 749
 CDnsPacket::UpdateRecords 749
 CEComDelegatingUnknown 413
 CEComDelegatingUnknown class 413
 ~CEComDelegatingUnknown 414
 CEComDelegatingUnknown 414
 CEComDelegatingUnknown::~CEComDelegatingUnknown 414
 CEComDelegatingUnknown::CEComDelegatingUnknown 414
 CEComUnknown 414
 CEComUnknown class 414
 ~CEComUnknown 416
 CEComUnknown 415
 InitializeInstance 416
 CEComUnknown::~CEComUnknown 416
 CEComUnknown::CEComUnknown 415
 CEComUnknown::InitializeInstance 416
 CEventDriven 762
 CEventDriven class 762
 ~CEventDriven 764
 Activate 764
 CEventDriven 764
 DisableCompletionDetection 765
 DisableEventsDetection 765
 EnableCompletionDetection 766
 EnableEventsDetection 766
 FinalizeAndReleaseA 766
 GetIEComUnknown 766
 GetServicingThread 767
 IsCurrentExecutionContext 767
 PostMessage 767
 RegisterRequestStatus 768
 RegisterSocket 768
 Release 768
 StartTimer 769
 StopAllTimers 770
 StopTimer 770
 UnregisterRequestStatus 770
 UnregisterSocket 771
 CEventDriven::~CEventDriven 764
 CEventDriven::Activate 764
 CEventDriven::CEventDriven 764
 CEventDriven::DisableCompletionDetection 765
 CEventDriven::DisableEventsDetection 765
 CEventDriven::EnableCompletionDetection 766
 CEventDriven::EnableEventsDetection 766
 CEventDriven::FinalizeAndReleaseA 766
 CEventDriven::GetIEComUnknown 766
 CEventDriven::GetServicingThread 767
 CEventDriven::IsCurrentExecutionContext 767
 CEventDriven::PostMessage 767
 CEventDriven::RegisterRequestStatus 768
 CEventDriven::RegisterSocket 768
 CEventDriven::Release 768
 CEventDriven::StartTimer 769
 CEventDriven::StopAllTimers 770
 CEventDriven::StopTimer 770
 CEventDriven::UnregisterRequestStatus 770
 CEventDriven::UnregisterSocket 771
 CExtendedKeyUsage 711
 CExtendedKeyUsage class 711
 != 714
 ~CExtendedKeyUsage 712
 = 714
 == 715
 CExtendedKeyUsage 712
 IsClientAuthenticationSet 712
 IsCodeSignedSet 713
 IsOcspSignedSet 713

IsServerAuthenticationSet 713
IsSmimeSet 713
IsTimestampingSet 714
CExtendedKeyUsage::!= 714
CExtendedKeyUsage::~CExtendedKeyUsage 712
CExtendedKeyUsage::= 714
CExtendedKeyUsage::== 715
CExtendedKeyUsage::CExtendedKeyUsage 712
CExtendedKeyUsage::IsClientAuthenticationSet 712
CExtendedKeyUsage::IsCodeSignedSet 713
CExtendedKeyUsage::IsOcspSignedSet 713
CExtendedKeyUsage::IsServerAuthenticationSet 713
CExtendedKeyUsage::IsSmimeSet 713
CExtendedKeyUsage::IsTimestampingSet 714
CFile 472
CFile class 472
 ~CFile 473
 CFile 473
 Close 474
 GetFileDescriptor 474
 Open 474
 Read 475
 Seek 476
 Stat 476
 Truncate 477
 Write 477, 478
CFile::~CFile 473
CFile::CFile 473
CFile::Close 474
CFile::EFlags 504
CFile::EFlags enumeration 504
CFile::EMode 505
CFile::EMode enumeration 505
CFile::ESeekPositions 505
CFile::ESeekPositions enumeration 505
CFile::GetFileDescriptor 474
CFile::Open 474
CFile::Read 475
CFile::Seek 476
CFile::SStat 511
CFile::SStat struct 511
CFile::Stat 476
CFile::Truncate 477
CFile::Write 477, 478
CFileTools 478
CFileTools class 478
 ChangeMode 479
 CreateNewDir 479
 Remove 480
 Rename 480
CFileTools::ChangeMode 479
CFileTools::CreateNewDir 479
CFileTools::Remove 480
CFileTools::Rename 480
CFqdn 527
CFqdn class 527
 != 530
 ~CFqdn 529
 = 530
 == 531
 CFqdn 528, 529
 GetFqdn 529
 GetPort 529
 m_strFqdn 528
 m_uPort 528
 Reset 529
 SetFqdn 530
 SetPort 530
CFqdn::!= 530
CFqdn::~CFqdn 529
CFqdn::= 530
CFqdn::== 531
CFqdn::CFqdn 528, 529
CFqdn::GetFqdn 529
CFqdn::GetPort 529
CFqdn::m_strFqdn 528
CFqdn::m_uPort 528
CFqdn::Reset 529
CFqdn::SetFqdn 530
CFqdn::SetPort 530
CFrameworkInitializer 789
CFrameworkInitializer class 789
 Finalize 790
 GetInitParameters 790
 GetInitParametersCount 791
 Initialize 791

InitializeSymbianNetwork 791	CHash::EAlgorithm 411
IsInitialized 791	CHash::EAlgorithm enumeration 411
CFrameworkInitializer::Finalize 790	CHash::End 353
CFrameworkInitializer::GetInitParameters 790	CHash::GetAlgorithm 354
CFrameworkInitializer::GetInitParametersCount 791	CHash::GetSizeInBits 354
CFrameworkInitializer::Initialize 791	CHash::GetSizeInBytes 354
CFrameworkInitializer::InitializeSymbianNetwork 791	CHash::SetState 354
CFrameworkInitializer::IsInitialized 791	CHash::Update 355
CFrameworkInitializer::SFrameworkFinalizeInfo 792	CHMac 355
CFrameworkInitializer::SFrameworkFinalizeInfo struct 792	CHMac class 355
~SFrameworkFinalizeInfo 793	~CHMac 356
m_astMemoryBlockInfo 792	Begin 356, 357
m_stMemoryStatistics 793	End 357, 358
m_uNumberOfAllocatedMemoryBlocks 793	GetAlgorithm 358
SFrameworkFinalizeInfo 793	GetSizeInBits 358
CFrameworkInitializer::SFrameworkFinalizeInfo::~SFrameworkFinalizeInfo 793	GetSizeInBytes 358
CFrameworkInitializer::SFrameworkFinalizeInfo::m_astMemoryBlockInfo 792	SetState 359
CFrameworkInitializer::SFrameworkFinalizeInfo::m_stMemoryStatistics 793	Update 359
CFrameworkInitializer::SFrameworkFinalizeInfo::m_uNumberOfAllocatedMemoryBlocks 793	CHMac::~CHMac 356
CFrameworkInitializer::SFrameworkFinalizeInfo::SFrameworkFinalizeInfo 793	CHMac::Begin 356, 357
CFrameworkInitializer::SFrameworkFinalizeInfo::SMemoryInfo 793	CHMac::EAlgorithm 411
CFrameworkInitializer::SFrameworkFinalizeInfo::SMemoryInfo struct 793	CHMac::EAlgorithm enumeration 411
CHash 351	CHMac::End 357, 358
CHash class 351	CHMac::GetAlgorithm 358
~CHash 352	CHMac::GetSizeInBits 358
Begin 353	CHMac::GetSizeInBytes 358
CHash 352	CHMac::SetState 359
End 353	CHMac::Update 359
GetAlgorithm 354	CHostFile 749
GetSizeInBits 354	CHostFile class 749
GetSizeInBytes 354	AddHostName 751
SetState 354	AddNamingAuthorityPointer 751
Update 355	AddService 752
CHash::~CHash 352	class CAsyncResolver 753
CHash::Begin 353	RemoveHostName 752
CHash::CHash 352	RemoveNamingAuthorityPointer 752
	RemoveService 753
	CHostFile::AddHostName 751
	CHostFile::AddNamingAuthorityPointer 751
	CHostFile::AddService 752
	CHostFile::RemoveHostName 752
	CHostFile::RemoveNamingAuthorityPointer 752
	CHostFile::RemoveService 753

ClssuerAlternateName 715
 ClssuerAlternateName class 715
 != 717
 ~ClssuerAlternateName 717
 = 717
 == 718
 ClssuerAlternateName 716, 717
 ClssuerAlternateName::!= 717
 ClssuerAlternateName::~ClssuerAlternateName 717
 ClssuerAlternateName::= 717
 ClssuerAlternateName::== 718
 ClssuerAlternateName::ClssuerAlternateName 716, 717
 CKerberos 426
 CKerberos class 426
 CKerberos 428
 GetGssContextAcceptor 429
 GetGssContextInitiator 429
 GetGssContextOptions 429
 GetPrincipal 430
 GetTicketA 430
 GetTicketOptions 430
 Initialize 431
 InitializeUser 431
 Release 432
 CKerberos::CKerberos 428
 CKerberos::GetGssContextAcceptor 429
 CKerberos::GetGssContextInitiator 429
 CKerberos::GetGssContextOptions 429
 CKerberos::GetPrincipal 430
 CKerberos::GetTicketA 430
 CKerberos::GetTicketOptions 430
 CKerberos::Initialize 431
 CKerberos::InitializeUser 431
 CKerberos::Release 432
 CKerberosDomainToRealm 432
 CKerberosDomainToRealm class 432
 ~CKerberosDomainToRealm 433
 CKerberosDomainToRealm 433
 GetDomain 433
 GetRealm 434
 SetDomain 434
 SetRealm 434
 CKerberosDomainToRealm::~CKerberosDomainToRealm 433
 CKerberosDomainToRealm::CKerberosDomainToRealm 433
 CKerberosDomainToRealm::GetDomain 433
 CKerberosDomainToRealm::GetRealm 434
 CKerberosDomainToRealm::SetDomain 434
 CKerberosDomainToRealm::SetRealm 434
 CKerberosGssContext 435
 CKerberosGssContext class 435
 ~CKerberosGssContext 435
 GetMix 436
 Release 436
 Unwrap 436
 VerifyMix 437
 Wrap 437
 CKerberosGssContext::~CKerberosGssContext 435
 CKerberosGssContext::GetMix 436
 CKerberosGssContext::Release 436
 CKerberosGssContext::Unwrap 436
 CKerberosGssContext::VerifyMix 437
 CKerberosGssContext::Wrap 437
 CKerberosGssContextAcceptor 438
 CKerberosGssContextAcceptor class 438
 Accept 439
 CKerberosGssContextAcceptor::Accept 439
 CKerberosGssContextInitiator 440
 CKerberosGssContextInitiator class 440
 Initiate 440
 CKerberosGssContextInitiator::Initiate 440
 CKerberosGssContextOptions 441
 CKerberosGssContextOptions class 441
 ~CKerberosGssContextOptions 442
 CloneFrom 442
 GetAnonymousity 443
 GetConfidentiality 443
 GetCredentialDelagation 443
 GetIntegrity 444
 GetMutualAuthentication 444
 GetOutOfSequenceDetection 444
 GetReplayDetection 445
 Release 445
 SetAnonymousity 445
 SetConfidentiality 445
 SetCredentialDelagation 446
 SetIntegrity 446

SetMutualAuthentication 446	CKerberosPrincipal::GetType 450
SetOutOfSequenceDetection 447	CKerberosPrincipal::Release 450
SetReplayDetection 447	CKerberosPrincipal::SetFullName 450
CKerberosGssContextOptions::~CKerberosGssContextOptions 442	CKerberosPrincipal::SetName 451
CKerberosGssContextOptions::CloneFrom 442	CKerberosPrincipal::SetNameComponents 451
CKerberosGssContextOptions::GetAnonymousity 443	CKerberosPrincipal::SetRealm 451
CKerberosGssContextOptions::GetConfidentiality 443	CKerberosPrincipal::SetType 452
CKerberosGssContextOptions::GetCredentialDelagation 443	CKerberosRealm 452
CKerberosGssContextOptions::GetIntegrity 444	CKerberosRealm class 452
CKerberosGssContextOptions::GetMutualAuthentication 444	~CKerberosRealm 453
CKerberosGssContextOptions::GetOutOfSequenceDetection 444	CKerberosRealm 453
CKerberosGssContextOptions::GetReplayDetection 445	GetKeyDistributionCenterFqdn 453
CKerberosGssContextOptions::Release 445	GetRealm 454
CKerberosGssContextOptions::SetAnonymousity 445	SetKeyDistributionCenterFqdn 454
CKerberosGssContextOptions::SetConfidentiality 445	SetRealm 454
CKerberosGssContextOptions::SetCredentialDelagation 446	CKerberosRealm::~CKerberosRealm 453
CKerberosGssContextOptions::SetIntegrity 446	CKerberosRealm::CKerberosRealm 453
CKerberosGssContextOptions::SetMutualAuthentication 446	CKerberosRealm::GetKeyDistributionCenterFqdn 453
CKerberosGssContextOptions::SetOutOfSequenceDetection 447	CKerberosRealm::GetRealm 454
CKerberosGssContextOptions::SetReplayDetection 447	CKerberosRealm::SetKeyDistributionCenterFqdn 454
CKerberosPrincipal 447	CKerberosRealm::SetRealm 454
CKerberosPrincipal class 447	CKerberosTicket 455
~CKerberosPrincipal 448	CKerberosTicket class 455
CloneFrom 448	~CKerberosTicket 455
GetFullName 449	GetClientPrincipal 456
GetName 449	GetServicePrincipal 456
GetNameComponents 449	Release 456
GetRealm 450	CKerberosTicket::~CKerberosTicket 455
GetType 450	CKerberosTicket::GetClientPrincipal 456
Release 450	CKerberosTicket::GetServicePrincipal 456
SetFullName 450	CKerberosTicket::Release 456
SetName 451	CKerberosTicketOptions 456
SetNameComponents 451	CKerberosTicketOptions class 456
SetRealm 451	~CKerberosTicketOptions 457
SetType 452	CloneFrom 457
CKerberosPrincipal::~CKerberosPrincipal 448	GetAddresses 458
CKerberosPrincipal::CloneFrom 448	GetForwardable 458
CKerberosPrincipal::EType 464	GetLifeTimeMs 458
CKerberosPrincipal::EType enumeration 464	GetProxiable 459
CKerberosPrincipal::GetFullName 449	GetRenewableLifeTimeMs 459
CKerberosPrincipal::GetName 449	Release 459
CKerberosPrincipal::GetNameComponents 449	SetAddresses 459
CKerberosPrincipal::GetRealm 450	SetForwardable 460

SetLifeTimeMs 460
 SetProxiable 460
 SetRenewableLifeTimeMs 460
 CKerberosTicketOptions::~CKerberosTicketOptions 457
 CKerberosTicketOptions::CloneFrom 457
 CKerberosTicketOptions::GetAddresses 458
 CKerberosTicketOptions::GetForwardable 458
 CKerberosTicketOptions::GetLifeTimeMs 458
 CKerberosTicketOptions::GetProxiable 459
 CKerberosTicketOptions::GetRenewableLifeTimeMs 459
 CKerberosTicketOptions::Release 459
 CKerberosTicketOptions::SetAddresses 459
 CKerberosTicketOptions::SetForwardable 460
 CKerberosTicketOptions::SetLifeTimeMs 460
 CKerberosTicketOptions::SetProxiable 460
 CKerberosTicketOptions::SetRenewableLifeTimeMs 460
 CKerberosTicketRequest 461
 CKerberosTicketRequest class 461
 Release 462
 CKerberosTicketRequest::Release 462
 CKeyUsage 718
 CKeyUsage class 718
 != 722
 ~CKeyUsage 720
 = 722
 == 723
 CKeyUsage 719
 IsCertificateSigningSet 720
 IsCrlSignedSet 720
 IsDataEnciphermentSet 720
 IsDecipherOnlySet 721
 IsDigitalSignatureSet 721
 IsEncipherOnlySet 721
 IsKeyAgreementSet 721
 IsKeyEnciphermentSet 722
 IsNonRepudiationSet 722
 CKeyUsage::!= 722
 CKeyUsage::~CKeyUsage 720
 CKeyUsage::= 722
 CKeyUsage::== 723
 CKeyUsage::CKeyUsage 719
 CKeyUsage::IsCertificateSigningSet 720
 CKeyUsage::IsCrlSignedSet 720
 CKeyUsage::IsDataEnciphermentSet 720
 CKeyUsage::IsDecipherOnlySet 721
 CKeyUsage::IsDigitalSignatureSet 721
 CKeyUsage::IsEncipherOnlySet 721
 CKeyUsage::IsKeyAgreementSet 721
 CKeyUsage::IsKeyEnciphermentSet 722
 CKeyUsage::IsNonRepudiationSet 722
 Classes 94, 318, 413, 426, 464, 522, 663, 738, 741, 762, 789, 794, 816, 839
 CList 170
 CList template 170
 != 183
 [] 184
 ~CList 173
 < 184
 <= 184
 = 184
 == 185
 > 185
 >= 185
 Allocate 174
 AllocateSorted 174
 Append 174
 CList 173
 Erase 175
 EraseAll 175
 EraseSorted 175
 Find 176
 FindPtrSorted 176
 FindSorted 176
 GetAt 177
 GetCapacity 177
 GetEndIndex 177
 GetFirstIndex 177
 GetLastIndex 178
 GetLockCapacity 178
 GetMaxElementIndex 178
 GetMinElementIndex 178
 GetSize 179
 Insert 179, 180
 InsertSorted 180
 IsEmpty 180
 IsFull 180

LockCapacity 181	CList::ReduceCapacity 181
Merge 181	CList::ReserveCapacity 181
ReduceCapacity 181	CList::SetComparisonFunction 182
ReserveCapacity 181	CList::Sort 182
SetComparisonFunction 182	CList::Split 182
Sort 182	CList::Swap 183
Split 182	CList::UnlockCapacity 183
Swap 183	CMac 531
UnlockCapacity 183	CMac class 531
CList::!= 183	!= 535
CList::[] 184	~CMac 532
CList::~CList 173	= 536
CList::< 184	== 536
CList::<= 184	CMac 532
CList::= 184	GetMac 533
CList::== 185	GetMacStr 533
CList::> 185	IsValid 534
CList::>= 185	SetMac 534, 535
CList::Allocate 174	SetWithDefaultMac 535
CList::AllocateSorted 174	CMac::!= 535
CList::Append 174	CMac::~CMac 532
CList::CList 173	CMac::= 536
CList::Erase 175	CMac::== 536
CList::EraseAll 175	CMac::CMac 532
CList::EraseSorted 175	CMac::GetMac 533
CList::Find 176	CMac::GetMacStr 533
CList::FindPtrSorted 176	CMac::IsValid 534
CList::FindSorted 176	CMac::SetMac 534, 535
CList::GetAt 177	CMac::SetWithDefaultMac 535
CList::GetCapacity 177	CMap 186
CList::GetEndIndex 177	CMap template 186
CList::GetFirstIndex 177	[] 195
CList::GetLastIndex 178	~CMap 189
CList::GetLockCapacity 178	= 196
CList::GetMaxElementIndex 178	Allocate 189
CList::GetMinElementIndex 178	CMap 189
CList::GetSize 179	Erase 190
CList::Insert 179, 180	EraseAll 190
CList::InsertSorted 180	EraseElement 190
CList::IsEmpty 180	FindPtr 190
CList::IsFull 180	GetAt 191
CList::LockCapacity 181	GetCapacity 191
CList::Merge 181	GetEndIndex 191

GetFirstIndex 191
 GetLastIndex 192
 GetLockCapacity 192
 GetMaxElementIndex 192
 GetMinElementIndex 192
 GetSize 193
 Insert 193
 IsEmpty 193
 IsFull 194
 LockCapacity 194
 ReduceCapacity 194
 ReserveCapacity 194
 SetComparisonFunction 195
 UnlockCapacity 195
 CMap::[] 195
 CMap::~CMap 189
 CMap::= 196
 CMap::Allocate 189
 CMap::CMap 189
 CMap::Erase 190
 CMap::EraseAll 190
 CMap::EraseElement 190
 CMap::FindPtr 190
 CMap::GetAt 191
 CMap::GetCapacity 191
 CMap::GetEndIndex 191
 CMap::GetFirstIndex 191
 CMap::GetLastIndex 192
 CMap::GetLockCapacity 192
 CMap::GetMaxElementIndex 192
 CMap::GetMinElementIndex 192
 CMap::GetSize 193
 CMap::Insert 193
 CMap::IsEmpty 193
 CMap::IsFull 194
 CMap::LockCapacity 194
 CMap::ReduceCapacity 194
 CMap::ReserveCapacity 194
 CMap::SetComparisonFunction 195
 CMap::UnlockCapacity 195
 CMapPair 196
 CMapPair template 196
 != 198
 ~CMapPair 198
 < 199
 CMapPair 197, 198
 GetFirst 198
 CMapPair::!= 198
 CMapPair::~CMapPair 198
 CMapPair::< 199
 CMapPair::CMapPair 197, 198
 CMapPair::GetFirst 198
 CMarshaler 117
 CMarshaler class 117
 ~CMarshaler 119
 << 120
 = 121
 >> 121
 Clear 119
 CMarshaler 119
 IsEmpty 119
 Load 120
 Store 120
 CMarshaler::~CMarshaler 119
 CMarshaler::<< 120
 CMarshaler::= 121
 CMarshaler::>> 121
 CMarshaler::Clear 119
 CMarshaler::CMarshaler 119
 CMarshaler::IsEmpty 119
 CMarshaler::Load 120
 CMarshaler::Store 120
 CMd5 360
 CMd5 class 360
 ~CMd5 361
 Begin 361
 CMd5 361
 End 362
 GetAlgorithm 362
 GetSizeInBits 362
 GetSizeInBytes 363
 SetState 363
 Update 363, 364
 CMd5::~CMd5 361
 CMd5::Begin 361
 CMd5::CMd5 361

CMd5::End 362	GetLineNumber 486
CMd5::GetAlgorithm 362	GetPointer 486
CMd5::GetSizeInBits 362	GetSize 486
CMd5::GetSizeInBytes 363	GetType 487
CMd5::SetState 363	IsPooled 487
CMd5::Update 363, 364	IsTemporarilyUntrackedFlagSet 487
CMd5Mac 364	SetPooled 487
CMd5Mac class 364	SetTemporarilyUntrackedFlag 487
~CMd5Mac 365	CMemoryAllocator::CMemoryBlock::GetFilename 486
Begin 365, 366	CMemoryAllocator::CMemoryBlock::GetLineNumber 486
CMd5Mac 365	CMemoryAllocator::CMemoryBlock::GetPointer 486
End 366	CMemoryAllocator::CMemoryBlock::GetSize 486
GetAlgorithm 366	CMemoryAllocator::CMemoryBlock::GetType 487
GetSizeInBits 367	CMemoryAllocator::CMemoryBlock::IsPooled 487
GetSizeInBytes 367	CMemoryAllocator::CMemoryBlock::IsTemporarilyUntrackedFlagSet
SetState 367	487
Update 367, 368	CMemoryAllocator::CMemoryBlock::SetPooled 487
CMd5Mac::~CMd5Mac 365	CMemoryAllocator::CMemoryBlock::SetTemporarilyUntrackedFlag
CMd5Mac::Begin 365, 366	487
CMd5Mac::CMd5Mac 365	CMemoryAllocator::Deallocate 482
CMd5Mac::End 366	CMemoryAllocator::DisableMemoryStatistics 482
CMd5Mac::GetAlgorithm 366	CMemoryAllocator::EnableMemoryStatistics 482
CMd5Mac::GetSizeInBits 367	CMemoryAllocator::EnumMemoryBlocks 483
CMd5Mac::GetSizeInBytes 367	CMemoryAllocator::GetMemoryBlock 483
CMd5Mac::SetState 367	CMemoryAllocator::GetMemoryStatistics 484
CMd5Mac::Update 367, 368	CMemoryAllocator::IMemoryBlockAccumulator 488
CMemoryAllocator 480	CMemoryAllocator::IMemoryBlockAccumulator class 488
CMemoryAllocator class 480	Accumulate 488
Allocate 481	CMemoryAllocator::IMemoryBlockAccumulator::Accumulate 488
Deallocate 482	CMemoryAllocator::PFNAllocateFunction 485
DisableMemoryStatistics 482	CMemoryAllocator::PFNDeallocateFunction 485
EnableMemoryStatistics 482	CMemoryAllocator::ResetMemoryStatistics 484
EnumMemoryBlocks 483	CMemoryAllocator::SetMemoryStatistics 484
GetMemoryBlock 483	CMemoryAllocator::SMemoryBlockExtraInfo 512
GetMemoryStatistics 484	CMemoryAllocator::SMemoryBlockExtraInfo struct 512
PFNAllocateFunction 485	m_pszFilename 512
PFNDeallocateFunction 485	m_pszType 512
ResetMemoryStatistics 484	m_uFlags 513
SetMemoryStatistics 484	m_uLineNumber 513
CMemoryAllocator::Allocate 481	SMemoryBlockExtraInfo 513
CMemoryAllocator::CMemoryBlock 485	CMemoryAllocator::SMemoryBlockExtraInfo::EMemoryBlockFlags
CMemoryAllocator::CMemoryBlock class 485	505
GetFilename 486	CMemoryAllocator::SMemoryBlockExtraInfo::EMemoryBlockFlags
	enumeration 505

CMemoryAllocator::SMemoryBlockExtraInfo::m_pszFilename 512	CMutex class 488
CMemoryAllocator::SMemoryBlockExtraInfo::m_pszType 512	~CMutex 489
CMemoryAllocator::SMemoryBlockExtraInfo::m_uFlags 513	CMutex 489
CMemoryAllocator::SMemoryBlockExtraInfo::m_uLineNumber 513	Lock 490
CMemoryAllocator::SMemoryBlockExtraInfo::SMemoryBlockExtraInfo 513	TryLock 490
	Unlock 490
CMemoryAllocator::SMemoryBlockHeader 513	CMutex::~CMutex 489
CMemoryAllocator::SMemoryBlockHeader struct 513	CMutex::CMutex 489
CMemoryAllocator::SMemoryPoolInfo 514	CMutex::Lock 490
CMemoryAllocator::SMemoryPoolInfo struct 514	CMutex::TryLock 490
CMemoryAllocator::SMemoryStatistics 514	CMutex::Unlock 490
CMemoryAllocator::SMemoryStatistics struct 514	CNetscapeCertificateType 723
CMemoryQueue 121	CNetscapeCertificateType class 723
CMemoryQueue class 121	 != 727
~CMemoryQueue 123	~CNetscapeCertificateType 725
Clear 123	 = 727
CMemoryQueue 123	 == 728
GetBlockSize 123	CNetscapeCertificateType 724, 725
Initialize 123	IsClientAuthenticationSet 725
IsBlockInQueue 124	IsObjectSignedCertificateAuthoritySet 726
IsEmpty 124	IsObjectSignedSet 726
Pop 124	IsServerAuthenticationSet 726
PopAbort 125	IsSmimeCertificateAuthoritySet 726
PopFinalize 125	IsSmimeSet 727
Push 125	IsSslCertificateAuthoritySet 727
PushAbort 125	CNetscapeCertificateType::!= 727
PushFinalize 126	CNetscapeCertificateType::~CNetscapeCertificateType 725
Uninitialize 126	CNetscapeCertificateType::= 727
CMemoryQueue::~CMemoryQueue 123	CNetscapeCertificateType::== 728
CMemoryQueue::Clear 123	CNetscapeCertificateType::CNetscapeCertificateType 724, 725
CMemoryQueue::CMemoryQueue 123	CNetscapeCertificateType::IsClientAuthenticationSet 725
CMemoryQueue::GetBlockSize 123	CNetscapeCertificateType::IsObjectSignedCertificateAuthoritySet 726
CMemoryQueue::Initialize 123	CNetscapeCertificateType::IsObjectSignedSet 726
CMemoryQueue::IsBlockInQueue 124	CNetscapeCertificateType::IsServerAuthenticationSet 726
CMemoryQueue::IsEmpty 124	CNetscapeCertificateType::IsSmimeCertificateAuthoritySet 726
CMemoryQueue::Pop 124	CNetscapeCertificateType::IsSmimeSet 727
CMemoryQueue::PopAbort 125	CNetscapeCertificateType::IsSslCertificateAuthoritySet 727
CMemoryQueue::PopFinalize 125	Config 261
CMemoryQueue::Push 125	Configuring Assertion Categories 272
CMemoryQueue::PushAbort 125	Configuring Tracing Levels 314
CMemoryQueue::PushFinalize 126	COsVersion 490
CMemoryQueue::Uninitialize 126	COsVersion class 490
CMutex 488	

-COsVersion	491	CPkcs12::Restore	730, 731
COsVersion	491	CPollRequestStatus	536
GetOsVersion	491	CPollRequestStatus class	536
COsVersion::~COsVersion	491	~CPollRequestStatus	538
COsVersion::COsVersion	491	CPollRequestStatus	538
COsVersion::EOsVersion	505	DisableCompletionDetection	538
COsVersion::EOsVersion enumeration	505	EnableCompletionDetection	538
COsVersion::GetOsVersion	491	GetCompletionDetectionState	539
CPair	199	GetRegisteredRequestStatusCount	539
CPair template	199	Poll	539
!= 202		RegisterRequestStatus	540
~CPair	201	UnregisterRequestStatus	540
< 202		CPollRequestStatus::~CPollRequestStatus	538
<= 202		CPollRequestStatus::CPollRequestStatus	538
= 202		CPollRequestStatus::DisableCompletionDetection	538
== 203		CPollRequestStatus::EnableCompletionDetection	538
> 203		CPollRequestStatus::GetCompletionDetectionState	539
>= 203		CPollRequestStatus::GetRegisteredRequestStatusCount	539
CPair	200, 201	CPollRequestStatus::Poll	539
GetFirst	201	CPollRequestStatus::RegisterRequestStatus	540
GetSecond	201	CPollRequestStatus::UnregisterRequestStatus	540
CPair::!=	202	CPollSocket	540
CPair::~CPair	201	CPollSocket class	540
CPair::<	202	~CPollSocket	542
CPair::<=	202	CPollSocket	541
CPair::=	202	DisableEventsDetection	542
CPair::==	203	EnableEventsDetection	542
CPair::>	203	GetEventsDetectionState	543
CPair::>=	203	GetRegisteredSocketCount	543
CPair::CPair	200, 201	Poll	543
CPair::GetFirst	201	RegisterSocket	544
CPair::GetSecond	201	UnregisterSocket	544
CPkcs12	728	CPollSocket::~CPollSocket	542
CPkcs12 class	728	CPollSocket::CPollSocket	541
~CPkcs12	730	DisableEventsDetection	542
= 732		EnableEventsDetection	542
CPkcs12	729	GetEventsDetectionState	543
GetCertificateChain	730	GetRegisteredSocketCount	543
Restore	730, 731	Poll	543
CPkcs12::~CPkcs12	730	RegisterSocket	544
CPkcs12::= 732		UnregisterSocket	544
CPkcs12::CPkcs12	729	CPool	204
CPkcs12::GetCertificateChain	730	CPool template	204

Allocate 205
Deallocate 205
Delete 205
GetCapacity 205
GetLockCapacity 205
Initialize 206
LockCapacity 206
New 206
ReduceCapacity 207
ReserveCapacity 207
Uninitialize 207
UnlockCapacity 208
CPool::Allocate 205
CPool::Deallocate 205
CPool::Delete 205
CPool::GetCapacity 205
CPool::GetLockCapacity 205
CPool::Initialize 206
CPool::LockCapacity 206
CPool::New 206
CPool::ReduceCapacity 207
CPool::ReserveCapacity 207
CPool::Uninitialize 207
CPool::UnlockCapacity 208
CPortableResolver 753
CPortableResolver class 753
 class CAsyncResolver 756
 EvAsyncClientSocketMgrBound 754
 EvAsyncClientSocketMgrConnected 754
 EvAsyncloSocketMgrReadyToRecv 755
 EvAsyncloSocketMgrReadyToSend 755
 EvAsyncSocketMgrClosed 755
 EvAsyncSocketMgrClosedByPeer 755
 EvAsyncSocketMgrErrorDetected 756
 QueryA 756
CPortableResolver::EvAsyncClientSocketMgrBound 754
CPortableResolver::EvAsyncClientSocketMgrConnected 754
CPortableResolver::EvAsyncloSocketMgrReadyToRecv 755
CPortableResolver::EvAsyncloSocketMgrReadyToSend 755
CPortableResolver::EvAsyncSocketMgrClosed 755
CPortableResolver::EvAsyncSocketMgrClosedByPeer 755
CPortableResolver::EvAsyncSocketMgrErrorDetected 756
CPortableResolver::QueryA 756
CPrivateKey 368
CPrivateKey class 368
 != 373
 ~CPrivateKey 370
 = 374
 == 374
 CPrivateKey 369, 370
 GetAlgorithm 370
 Restore 370, 371
 RestoreDer 371
 RestorePem 372
 Store 372
 StoreDer 373
 StorePem 373
CPrivateKey crypto algorithm configuration macros 284
CPrivateKey::!= 373
CPrivateKey::~CPrivateKey 370
CPrivateKey::= 374
CPrivateKey::== 374
CPrivateKey::CPrivateKey 369, 370
CPrivateKey::GetAlgorithm 370
CPrivateKey::Restore 370, 371
CPrivateKey::RestoreDer 371
CPrivateKey::RestorePem 372
CPrivateKey::Store 372
CPrivateKey::StoreDer 373
CPrivateKey::StorePem 373
CPublicKey 374
CPublicKey class 374
 != 379
 ~CPublicKey 376
 = 379
 == 379
 CPublicKey 375, 376
 GetAlgorithm 376
 Restore 376
 RestoreDer 377
 RestorePem 377
 Store 377
 StoreDer 378
 StorePem 378
CPublicKey crypto algorithm configuration macros 284
CPublicKey::!= 379

CPublicKey::~CPublicKey 376
CPublicKey::= 379
CPublicKey::== 379
CPublicKey::CPublicKey 375, 376
CPublicKey::GetAlgorithm 376
CPublicKey::Restore 376
CPublicKey::RestoreDer 377
CPublicKey::RestorePem 377
CPublicKey::Store 377
CPublicKey::StoreDer 378
CPublicKey::StorePem 378
CQueue 208
CQueue template 208
!= 213
~CQueue 210
< 214
<= 214
= 214
== 214
> 215
>= 215
CQueue 209, 210
Dequeue 210
Enqueue 210
EraseAll 211
GetCapacity 211
GetFront 211
GetLockCapacity 211
GetSize 211
IsEmpty 212
IsFull 212
LockCapacity 212
ReduceCapacity 212
ReserveCapacity 213
UnlockCapacity 213
CQueue::!= 213
CQueue::~CQueue 210
CQueue::< 214
CQueue::<= 214
CQueue::= 214
CQueue::== 214
CQueue::> 215
CQueue::>= 215
CQueue::CQueue 209, 210
CQueue::Dequeue 210
CQueue::Enqueue 210
CQueue::EraseAll 211
CQueue::GetCapacity 211
CQueue::GetFront 211
CQueue::GetLockCapacity 211
CQueue::GetSize 211
CQueue::IsEmpty 212
CQueue::IsFull 212
CQueue::LockCapacity 212
CQueue::ReduceCapacity 212
CQueue::ReserveCapacity 213
CQueue::UnlockCapacity 213
CreateEComInstance 420, 421
CreateEComInstance function 420, 421
CRegExp 738
CRegExp class 738
 Find 739
 FindAndReplace 739
 FindSubExpressions 740
CRegExp::Find 739
CRegExp::FindAndReplace 739
CRegExp::FindSubExpressions 740
CRegExp::SSubExpression 740
CRegExp::SSubExpression struct 740
CResolver 756
CResolver class 756
 GetEnumUris 757
 GetHostByAddress 757
 GetHostByName 757
 GetNameServers 758
 GetNamingAuthorityPointers 758
 GetServices 758
 SetNameServers 758
CResolver::GetEnumUris 757
CResolver::GetHostByAddress 757
CResolver::GetHostByName 757
CResolver::GetNameServers 758
CResolver::GetNamingAuthorityPointers 758
CResolver::GetServices 758
CResolver::SetNameServers 758
CResolverCache 759

CRResolverCache class 759
 class CAsyncResolver 760
 Compare 759
CRResolverCache::Compare 759
CRsa 379
CRsa class 379
 ~CRsa 381
 CRsa 381
 GenerateKey 381
 GetPrivateKey 382
 GetPublicKey 382
 PrivateKeyDecrypt 382, 383
 PrivateKeyEncrypt 383, 384
 PublicKeyDecrypt 384, 385
 PublicKeyEncrypt 386
 SetKey 387
 Sign 387
 Verify 388
CRsa::~CRsa 381
CRsa::CRsa 381
CRsa::GenerateKey 381
CRsa::GetPrivateKey 382
CRsa::GetPublicKey 382
CRsa::PrivateKeyDecrypt 382, 383
CRsa::PrivateKeyEncrypt 383, 384
CRsa::PublicKeyDecrypt 384, 385
CRsa::PublicKeyEncrypt 386
CRsa::SetKey 387
CRsa::Sign 387
CRsa::Verify 388
Crypto 318
Crypto engine configuration macros 281
CSecurePrng 389
CSecurePrng class 389
 Generate 389, 390
 SetSeed 390
CSecurePrng::Generate 389, 390
CSecurePrng::SetSeed 390
CSecureSeed 390
CSecureSeed class 390
 class Foo 391
 GenerateSeed 391
CSecureSeed::GenerateSeed 391
CServicingThread 771
CServicingThread class 771
 ~CServicingThread 774
 CreateInstance 774
 CServicingThread 774
 InitializeInstance 775
 NonDelegatingQueryIf 775
 NonDelegatingReleaseIfRef 775
 UninitializeInstance 775
CServicingThread::~CServicingThread 774
CServicingThread::CreateInstance 774
CServicingThread::CServicingThread 774
CServicingThread::InitializeInstance 775
CServicingThread::NonDelegatingQueryIf 775
CServicingThread::NonDelegatingReleaseIfRef 775
CServicingThread::UninitializeInstance 775
CSha1 391
CSha1 class 391
 ~CSha1 393
 Begin 393
 CSha1 392
 End 393
 GetAlgorithm 394
 GetSizeInBits 394
 GetSizeInBytes 394
 SetState 394
 Update 395
CSha1::~CSha1 393
CSha1::Begin 393
CSha1::CSha1 392
CSha1::End 393
CSha1::GetAlgorithm 394
CSha1::GetSizeInBits 394
CSha1::GetSizeInBytes 394
CSha1::SetState 394
CSha1::Update 395
CSha1Mac 395
CSha1Mac class 395
 ~CSha1Mac 396
 Begin 397
 CSha1Mac 396
 End 397
 GetAlgorithm 398

GetSizeInBits 398
GetSizeInBytes 398
SetState 398
Update 399
CSha1Mac::~CSha1Mac 396
CSha1Mac::Begin 397
CSha1Mac::CSha1Mac 396
CSha1Mac::End 397
CSha1Mac::GetAlgorithm 398
CSha1Mac::GetSizeInBits 398
CSha1Mac::GetSizeInBytes 398
CSha1Mac::SetState 398
CSha1Mac::Update 399
CSha2 399
CSha2 class 399
~CSha2 401
Begin 401
CSha2 401
End 401, 402
GetAlgorithm 402
GetSizeInBits 402
GetSizeInBytes 402
SetDefaultAlgorithm 403
SetState 403
Update 403, 404
CSha2::~CSha2 401
CSha2::Begin 401
CSha2::CSha2 401
CSha2::End 401, 402
CSha2::GetAlgorithm 402
CSha2::GetSizeInBits 402
CSha2::GetSizeInBytes 402
CSha2::SetDefaultAlgorithm 403
CSha2::SetState 403
CSha2::Update 403, 404
CSha2Mac 404
CSha2Mac class 404
~CSha2Mac 405
Begin 405
CSha2Mac 405
End 406
GetAlgorithm 406
GetSizeInBits 406
GetSizeInBytes 407
SetDefaultAlgorithm 407
SetState 407
Update 407, 408
CSha2Mac::~CSha2Mac 405
CSha2Mac::Begin 405
CSha2Mac::CSha2Mac 405
CSha2Mac::End 406
CSha2Mac::GetAlgorithm 406
CSha2Mac::GetSizeInBits 406
CSha2Mac::GetSizeInBytes 407
CSha2Mac::SetDefaultAlgorithm 407
CSha2Mac::SetState 407
CSha2Mac::Update 407, 408
CSharedPtr 75
CSharedPtr template 75
!= 77, 78
* 78
_Type** 79
~CSharedPtr 77
< 79
<= 80
= 81
== 81, 82
> 82, 83
-> 83
>= 83, 84
CSharedPtr 76
Get 77
Reset 77
CSharedPtr::!= 77, 78
CSharedPtr::* 78
CSharedPtr::_Type** 79
CSharedPtr::~CSharedPtr 77
CSharedPtr::< 79
CSharedPtr::<= 80
CSharedPtr::= 81
CSharedPtr::== 81, 82
CSharedPtr::> 82, 83
CSharedPtr::-> 83
CSharedPtr::>= 83, 84
CSharedPtr::CSharedPtr 76
CSharedPtr::Get 77

CSharedPtr::Reset	77	IsEqualPort	551
CSntpClient	816	IsEqualScope	552
CSntpClient class	816	IsEqualScopeId	552
~CSntpClient	818	IsInet6AddressGlobal	552
Activate	818	IsInet6AddressLinkLocal	552
ClearServerList	819	IsInet6AddressLoopback	553
CSntpClient	818	IsInet6AddressMulticast	553
GetTimeS	819	IsInet6AddressSiteLocal	553
InsertServer	819	IsInet6AddressUniqueLocal	553
QueryServer	820, 821	IsInet6AddressUnspecified	553
SetSntpPacketSentToServer	822	IsInet6AddressV4Mapped	554
SetTimeoutMs	822	IsInetAddressLoopback	554
Terminate	822	IsInetAddressMulticast	554
CSntpClient::~CSntpClient	818	IsValid	554
CSntpClient::Activate	818	IsValidAddress	554
CSntpClient::ClearServerList	819	IsValidFamily	555
CSntpClient::CSntpClient	818	IsValidPort	555
CSntpClient::GetTimeS	819	Reset	555
CSntpClient::InsertServer	819	SetAddress	555, 556
CSntpClient::QueryServer	820, 821	SetPort	557
CSntpClient::SetSntpPacketSentToServer	822	SetScopeId	557
CSntpClient::SetTimeoutMs	822	sockaddr	558
CSntpClient::Terminate	822	sockaddr*	558
CSocket	544	TInetAddr*	559
CSocket class	544	CSocketAddr::!=	557
CSocketAddr	545	CSocketAddr::~CSocketAddr	547
CSocketAddr class	545	CSocketAddr::=	557
!=	557	CSocketAddr::==	558
~CSocketAddr	547	CSocketAddr::const sockaddr*	558
=	557	CSocketAddr::CSocketAddr	546, 547
==	558	CSocketAddr::EAddressFamily	651
const sockaddr*	558	CSocketAddr::EAddressFamily enumeration	651
CSocketAddr	546, 547	CSocketAddr::EStandardAddress	651
GetAddress	548, 549	CSocketAddr::EStandardAddress enumeration	651
GetFamily	549	CSocketAddr::GetAddress	548, 549
GetPort	549	CSocketAddr::GetFamily	549
GetScopeId	550	CSocketAddr::GetPort	549
GetSize	550	CSocketAddr::GetScopeId	550
Inet6AnyAddress	550	CSocketAddr::GetSize	550
InetAnyAddress	550	CSocketAddr::Inet6AnyAddress	550
InetBroadcastAddress	551	CSocketAddr::InetAnyAddress	550
IsEqualAddress	551	CSocketAddr::InetBroadcastAddress	551
IsEqualFamily	551	CSocketAddr::IsEqualAddress	551

CSocketAddr::IsEqualFamily 551
 CSocketAddr::IsEqualPort 551
 CSocketAddr::IsEqualScope 552
 CSocketAddr::IsEqualScopId 552
 CSocketAddr::IsInet6AddressGlobal 552
 CSocketAddr::IsInet6AddressLinkLocal 552
 CSocketAddr::IsInet6AddressLoopback 553
 CSocketAddr::IsInet6AddressMulticast 553
 CSocketAddr::IsInet6AddressSiteLocal 553
 CSocketAddr::IsInet6AddressUniqueLocal 553
 CSocketAddr::IsInet6AddressUnspecified 553
 CSocketAddr::IsInet6AddressV4Mapped 554
 CSocketAddr::IsInetAddressLoopback 554
 CSocketAddr::IsInetAddressMulticast 554
 CSocketAddr::IsValid 554
 CSocketAddr::IsValidAddress 554
 CSocketAddr::IsValidFamily 555
 CSocketAddr::IsValidPort 555
 CSocketAddr::Reset 555
 CSocketAddr::SetAddress 555, 556
 CSocketAddr::SetPort 557
 CSocketAddr::SetScopId 557
 CSocketAddr::sockaddr 558
 CSocketAddr::sockaddr* 558
 CSocketAddr::TInetAddr* 559
 CStack 215
 CStack template 215
 != 221
 ~CStack 217
 < 221
 <= 222
 = 222
 == 222
 > 222
 >= 223
 CStack 217
 EraseAll 218
 GetCapacity 218
 GetLockCapacity 218
 GetSize 218
 GetTop 219
 IsEmpty 219
 IsFull 219
 LockCapacity 219
 Pop 220
 Push 220
 ReduceCapacity 220
 ReserveCapacity 220
 UnlockCapacity 221
 CStack::!= 221
 CStack::~CStack 217
 CStack::< 221
 CStack::<= 222
 CStack::= 222
 CStack::== 222
 CStack::> 222
 CStack::>= 223
 CStack::CStack 217
 CStack::EraseAll 218
 CStack::GetCapacity 218
 CStack::GetLockCapacity 218
 CStack::GetSize 218
 CStack::GetTop 219
 CStack::IsEmpty 219
 CStack::IsFull 219
 CStack::LockCapacity 219
 CStack::Pop 220
 CStack::Push 220
 CStack::ReduceCapacity 220
 CStack::ReserveCapacity 220
 CStack::UnlockCapacity 221
 CString 126
 CString class 126
 != 139
 [] 139
 ~CString 131
 + 140
 += 141
 < 142
 <= 142, 143
 = 143, 144
 == 144
 > 145
 >= 145, 146
 Append 131
 CaseInsCmp 131, 132

CStr 132
CString 130
Erase 132
EraseAll 133
FindSubstring 133
Format 134
FormatV 134
GetAt 134
GetBuffer 134
GetCapacity 134
GetSize 135
Insert 135, 136
IsEmpty 136
ReduceCapacity 136
ReserveCapacity 137
Resize 137
SetAt 137
ToLowerCase 138
ToUpperCase 138
TrimBothSide 138
TrimLeftSide 138
TrimRightSide 138
CString::!= 139
CString::[] 139
CString::~CString 131
CString::+ 140
CString::+= 141
CString::< 142
CString::<= 142, 143
CString::= 143, 144
CString::== 144
CString::> 145
CString::>= 145, 146
CString::Append 131
CString::CaseInsCmp 131, 132
CString::CStr 132
CString::CString 130
CString::Erase 132
CString::EraseAll 133
CString::FindSubstring 133
CString::Format 134
CString::FormatV 134
CString::GetAt 134
CString::GetBuffer 134
CString::GetCapacity 134
CString::GetSize 135
CString::Insert 135, 136
CString::IsEmpty 136
CString::ReduceCapacity 136
CString::ReserveCapacity 137
CString::Resize 137
CString::SetAt 137
CString::ToLowerCase 138
CString::ToUpperCase 138
CString::TrimBothSide 138
CString::TrimLeftSide 138
CString::TrimRightSide 138
CSubAllocator 146
CSubAllocator class 146
 ~CSubAllocator 148
 Allocate 148
 CSubAllocator 147
 GetTotalAllocatedBytes 148
 GetTotalAvailableBytes 148
 GetTotalNumAllocations 148
 GetTotalNumReleases 149
 Release 149
 Renew 149
CSubAllocator::~CSubAllocator 148
CSubAllocator::Allocate 148
CSubAllocator::CSubAllocator 147
CSubAllocator::GetTotalAllocatedBytes 148
CSubAllocator::GetTotalAvailableBytes 148
CSubAllocator::GetTotalNumAllocations 148
CSubAllocator::GetTotalNumReleases 149
CSubAllocator::Release 149
CSubAllocator::Renew 149
CSubjectAlternateName 732
CSubjectAlternateName class 732
 != 734
 ~CSubjectAlternateName 734
 = 734
 == 735
 CSubjectAlternateName 733, 734
 CSubjectAlternateName::!= 734
 CSubjectAlternateName::~CSubjectAlternateName 734

CSubjectAlternateName::= 734	CTcpServerSocket::CTcpServerSocket 560
CSubjectAlternateName::= 735	CTcpServerSocket::GetAddressFamily 562
CSubjectAlternateName::CSubjectAlternateName 733, 734	CTcpServerSocket::GetHandle 562
CSubjectKeyIdentifier 735	CTcpServerSocket::GetLocalAddress 563
CSubjectKeyIdentifier class 735	CTcpServerSocket::GetProtocolFamily 563
!= 737	CTcpServerSocket::GetSocketType 563
~CSubjectKeyIdentifier 737	CTcpServerSocket::Listen 564
= 737	CTcpServerSocket::Release 564
== 738	CTcpServerSocket::Set8021QUserPriority 564
CSubjectKeyIdentifier 736	CTcpServerSocket::SetBlocking 565
GetSubjectKeyIdentifier 737	CTcpServerSocket::SetIpv6UnicastHops 565
CSubjectKeyIdentifier::!= 737	CTcpServerSocket::SetKeepAlive 565
CSubjectKeyIdentifier::~CSubjectKeyIdentifier 737	CTcpServerSocket::SetReuseAddress 565
CSubjectKeyIdentifier::= 737	CTcpServerSocket::SetTos 566
CSubjectKeyIdentifier::== 738	CTcpServerSocket::SetWindowsReceivingFlowspec 566
CSubjectKeyIdentifier::CSubjectKeyIdentifier 736	CTcpServerSocket::SetWindowsSendingFlowspec 566
CSubjectKeyIdentifier::GetSubjectKeyIdentifier 737	CTcpSocket 567
CTcpServerSocket 559	CTcpSocket class 567
CTcpServerSocket class 559	Bind 568
Accept 560	Close 568
Bind 561	Connect 569
Close 561	Create 569
Create 562	CTcpSocket 568
CTcpServerSocket 560	GetAddressFamily 570
GetAddressFamily 562	GetHandle 570
GetHandle 562	GetLocalAddress 570
GetLocalAddress 563	GetPeerAddress 571
GetProtocolFamily 563	GetProtocolFamily 571
GetSocketType 563	GetSocketType 571
Listen 564	Recv 572
Release 564	RecvFrom 573
Set8021QUserPriority 564	Release 574
SetBlocking 565	Send 574, 575
SetIpv6UnicastHops 565	SendTo 575, 576
SetKeepAlive 565	Set8021QUserPriority 576
SetReuseAddress 565	SetBlocking 576
SetTos 566	SetIpv6UnicastHops 577
SetWindowsReceivingFlowspec 566	SetKeepAlive 577
SetWindowsSendingFlowspec 566	SetNagle 577
CTcpServerSocket::Accept 560	SetReceiveBufferSize 578
CTcpServerSocket::Bind 561	SetReuseAddress 578
CTcpServerSocket::Close 561	SetTos 578
CTcpServerSocket::Create 562	SetTransmitBufferSize 579

SetWindowsReceivingFlowspec 579	CTcpSocketOptions::ApplyOptions 581
SetWindowsSendingFlowspec 579	CTcpSocketOptions::CreateInstance 581
CTcpSocket::Bind 568	CTcpSocketOptions::NonDelegatingQueryIf 582
CTcpSocket::Close 568	CTcpSocketOptions::Set8021QUserPriority 582
CTcpSocket::Connect 569	CTcpSocketOptions::SetConnectTimeoutMs 582
CTcpSocket::Create 569	CTcpSocketOptions::SetKeepAlive 582
CTcpSocket::CTcpSocket 568	CTcpSocketOptions::SetNagle 583
CTcpSocket::GetAddressFamily 570	CTcpSocketOptions::SetReceiveBufferSize 583
CTcpSocket::GetHandle 570	CTcpSocketOptions::SetTos 583
CTcpSocket::GetLocalAddress 570	CTcpSocketOptions::SetTransmitBufferSize 584
CTcpSocket::GetPeerAddress 571	CTcpSocketOptions::SetWindowsReceivingFlowspec 584
CTcpSocket::GetProtocolFamily 571	CTcpSocketOptions::SetWindowsSendingFlowspec 584
CTcpSocket::GetSocketType 571	CThread 491
CTcpSocket::Recv 572	CThread class 491
CTcpSocket::RecvFrom 573	!= 500
CTcpSocket::Release 574	~CThread 493
CTcpSocket::Send 574, 575	== 500
CTcpSocket::SendTo 575, 576	CreateKey 493
CTcpSocket::Set8021QUserPriority 576	CThread 493
CTcpSocket::SetBlocking 576	DeleteKey 494
CTcpSocket::SetIpv6UnicastHops 577	FinalizeTsd 494
CTcpSocket::SetKeepAlive 577	GetCurrentId 494
CTcpSocket::SetNagle 577	GetId 495
CTcpSocket::SetReceiveBufferSize 578	GetName 495
CTcpSocket::SetReuseAddress 578	GetNativeThreadId 495
CTcpSocket::SetTos 578	GetSpecific 495
CTcpSocket::SetTransmitBufferSize 579	GetState 496
CTcpSocket::SetWindowsReceivingFlowspec 579	GetThreadSelf 496
CTcpSocket::SetWindowsSendingFlowspec 579	GetThreadStackInfo 497
CTcpSocketOptions 579	InitializeTsd 498
CTcpSocketOptions class 579	IsCurrentThread 498
ApplyOptions 581	Join 498
CreateInstance 581	mxt_tsdk 501
NonDelegatingQueryIf 582	PFNEntryPoint 501
Set8021QUserPriority 582	PFNKeyDeletion 501
SetConnectTimeoutMs 582	ResetThreadStackInfo 499
SetKeepAlive 582	SetSpecific 499
SetNagle 583	StartThread 499
SetReceiveBufferSize 583	CThread::!= 500
SetTos 583	CThread::~CThread 493
SetTransmitBufferSize 584	CThread::== 500
SetWindowsReceivingFlowspec 584	CThread::CreateKey 493
SetWindowsSendingFlowspec 584	CThread::CThread 493

CThread::DeleteKey 494
CThread::EPriority 506
CThread::EPriority enumeration 506
CThread::FinalizeTsd 494
CThread::GetCurrentId 494
CThread::GetId 495
CThread::GetName 495
CThread::GetNativeThreadId 495
CThread::GetSpecific 495
CThread::GetState 496
CThread::GetThreadSelf 496
CThread::GetThreadStackInfo 497
CThread::InitializeTsd 498
CThread::IsCurrentThread 498
CThread::Join 498
CThread::mxt_tskey 501
CThread::PFNEntryPoint 501
CThread::PFNKeyDeletion 501
CThread::ResetThreadStackInfo 499
CThread::SetSpecific 499
CThread::StartThread 499
CThread::SThreadStackInfo 514
CThread::SThreadStackInfo struct 514
CTime 822
CTime class 822
!= 830
~CTime 824
< 830
<= 830
= 831
== 831
> 831
>= 832
CTime 824
GetDate 824
GetDateAndTime 825
GetDayOfWeek 825
GetGmtTime 825
GetGregorianDate 825
GetJulianDate 826
GetJulianDateAndTime 826
GetSystemTimeZone 826
GetTime 827
IsDayLightSavingInEffect 827
IsLeapYear 827
PinTime 827
SetDate 828
SetDateAndTime 828
SetJulianDateAndTime 828
SetSystemTime 829
SetSystemTimeZone 829
SetTime 829
SetTimeZone 830
CTime::!= 830
CTime::~CTime 824
CTime::< 830
CTime::<= 830
CTime::= 831
CTime::== 831
CTime::> 831
CTime::>= 832
CTime::CTime 824
CTime::EDayOfWeek 838
CTime::EDayOfWeek enumeration 838
CTime::GetDate 824
CTime::GetDateAndTime 825
CTime::GetDayOfWeek 825
CTime::GetGmtTime 825
CTime::GetGregorianDate 825
CTime::GetJulianDate 826
CTime::GetJulianDateAndTime 826
CTime::GetSystemTimeZone 826
CTime::GetTime 827
CTime::IsDayLightSavingInEffect 827
CTime::IsLeapYear 827
CTime::PinTime 827
CTime::SetDate 828
CTime::SetDateAndTime 828
CTime::SetJulianDateAndTime 828
CTime::SetSystemTime 829
CTime::SetSystemTimeZone 829
CTime::SetTime 829
CTime::SetTimeZone 830
CTimer 501
CTimer class 501
~CTimer 502

CTimer 502
CyclicWait 502
GetSystemUpTimeMs 503
StartCyclicWait 503
StopCyclicWait 503
Wait 503
CTimer::~CTimer 502
CTimer::CTimer 502
CTimer::CyclicWait 502
CTimer::GetSystemUpTimeMs 503
CTimer::StartCyclicWait 503
CTimer::StopCyclicWait 503
CTimer::Wait 503
CTimeZone 832
CTimeZone class 832
~CTimeZone 834
= 837
ConvertFromLocaleToUTC 834
ConvertFromUTCToLocale 835
CTimeZone 833, 834
GetTimeZone 835
IsDayLightSavingInEffect 836
IsLeapYear 836
IsValid 837
SetTimeZone 837
CTimeZone::~CTimeZone 834
CTimeZone::= 837
CTimeZone::ConvertFromLocaleToUTC 834
CTimeZone::ConvertFromUTCToLocale 835
CTimeZone::CTimeZone 833, 834
CTimeZone::GetTimeZone 835
CTimeZone::IsDayLightSavingInEffect 836
CTimeZone::IsLeapYear 836
CTimeZone::IsValid 837
CTimeZone::SetTimeZone 837
CTls 796
CTls class 796
 Initialize 796
 Instance 796
 Uninitialize 797
CTls::Initialize 796
CTls::Instance 796
CTls::Uninitialize 797
CTlsContext 797
CTlsContext class 797
 ~CTlsContext 799
 = 803
 CTlsContext 798, 799
 GetCertificateChain 799
 GetCiphers 799
 GetEphemeralDiffieHellman 800
 GetPeerAuthentication 800
 GetProtocolVersions 800
 GetTrustedCertificates 800
 SetCertificateChain 800
 SetCiphers 801
 SetEphemeralDiffieHellman 802
 SetPeerAuthentication 802
 SetProtocolVersions 802
 SetTrustedCertificates 802
 CTlsContext::~CTlsContext 799
 CTlsContext::= 803
 CTlsContext::CTlsContext 798, 799
 CTlsContext::GetCertificateChain 799
 CTlsContext::GetCiphers 799
 CTlsContext::GetEphemeralDiffieHellman 800
 CTlsContext::GetPeerAuthentication 800
 CTlsContext::GetProtocolVersions 800
 CTlsContext::GetTrustedCertificates 800
 CTlsContext::SetCertificateChain 800
 CTlsContext::SetCiphers 801
 CTlsContext::SetEphemeralDiffieHellman 802
 CTlsContext::SetPeerAuthentication 802
 CTlsContext::SetProtocolVersions 802
 CTlsContext::SetTrustedCertificates 802
 CTlsSession 803
 CTlsSession class 803
 ~CTlsSession 805
 = 806
 CTlsSession 804
 GetId 805
 Restore 805
 Store 805
 CTlsSession::~CTlsSession 805
 CTlsSession::= 806
 CTlsSession::CTlsSession 804

CTIsSession::GetId	805	CUdpSocket::Recv	589, 590
CTIsSession::Restore	805	CUdpSocket::RecvFrom	590, 591
CTIsSession::Store	805	CUdpSocket::Release	591
CUdpSocket	585	CUdpSocket::Send	592
CUdpSocket class	585	CUdpSocket::SendTo	593
Bind	586	CUdpSocket::Set8021QUserPriority	594
Close	586	CUdpSocket::SetAllowAnySource	594
Connect	587	CUdpSocket::SetBlocking	594
Create	587	CUdpSocket::SetBroadcast	595
CUdpSocket	586	CUdpSocket::SetIpv6UnicastHops	595
GetAddressFamily	588	CUdpSocket::SetReceiveBufferSize	595
GetHandle	588	CUdpSocket::SetTos	596
GetLocalAddress	588	CUdpSocket::SetTransmitBufferSize	596
GetPeerAddress	588	CUdpSocket::SetUdpChecksum	596
GetProtocolFamily	589	CUdpSocket::SetWindowsReceivingFlowspec	596
GetSocketType	589	CUdpSocket::SetWindowsSendingFlowspec	597
Recv	589, 590	CUdpTracing	597
RecvFrom	590, 591	CUdpTracing class	597
Release	591	Finalize	598
Send	592	Initialize	598
SendTo	593	CUdpTracing::Finalize	598
Set8021QUserPriority	594	CUdpTracing::Initialize	598
SetAllowAnySource	594	CUncmp	223
SetBlocking	594	CUncmp template	223
SetBroadcast	595	!=	226
SetIpv6UnicastHops	595	~CUncmp	226
SetReceiveBufferSize	595	<	226
SetTos	596	CUncmp	225, 226
SetTransmitBufferSize	596	CUncmp::!=	226
SetUdpChecksum	596	CUncmp::~CUncmp	226
SetWindowsReceivingFlowspec	596	CUncmp::<	226
SetWindowsSendingFlowspec	597	CUncmp::CUncmp	225, 226
CUdpSocket::Bind	586	CVector	227
CUdpSocket::Close	586	CVector template	227
CUdpSocket::Connect	587	!=	241
CUdpSocket::Create	587	[]	242
CUdpSocket::CUdpSocket	586	~CVector	231
CUdpSocket::GetAddressFamily	588	<	242
CUdpSocket::GetHandle	588	<=	243
CUdpSocket::GetLocalAddress	588	=	243
CUdpSocket::GetPeerAddress	588	==	243
CUdpSocket::GetProtocolFamily	589	>	243
CUdpSocket::GetSocketType	589	>=	244

Allocate 232	CVector::Append 232
AllocateSorted 232	CVector::CVector 231
Append 232	CVector::Erase 233
CVector 231	CVector::EraseAll 233
Erase 233	CVector::EraseSorted 233
EraseAll 233	CVector::Find 234
EraseSorted 233	CVector::FindPtrSorted 234
Find 234	CVector::FindSorted 234
FindPtrSorted 234	CVector::GetAt 235
FindSorted 234	CVector::GetCapacity 235
GetAt 235	CVector::GetEndIndex 235
GetCapacity 235	CVector::GetFirstIndex 235
GetEndIndex 235	CVector::GetLastIndex 236
GetFirstIndex 235	CVector::GetLockCapacity 236
GetLastIndex 236	CVector::GetMaxElementIndex 236
GetLockCapacity 236	CVector::GetMinElementIndex 236
GetMaxElementIndex 236	CVector::GetSize 237
GetMinElementIndex 236	CVector::Insert 237, 238
GetSize 237	CVector::InsertSorted 238
Insert 237, 238	CVector::IsEmpty 238
InsertSorted 238	CVector::IsFull 239
IsEmpty 238	CVector::LockCapacity 239
IsFull 239	CVector::Merge 239
LockCapacity 239	CVector::ReduceCapacity 239
Merge 239	CVector::ReserveCapacity 240
ReduceCapacity 239	CVector::SetComparisonFunction 240
ReserveCapacity 240	CVector::Sort 240
SetComparisonFunction 240	CVector::Split 240
Sort 240	CVector::Swap 241
Split 240	CVector::UnlockCapacity 241
Swap 241	CVersion 149
UnlockCapacity 241	CVersion class 149
CVector::!= 241	!= 154
CVector::[] 242	~CVersion 151
CVector::~CVector 231	< 154
CVector::< 242	<= 155
CVector::<= 243	= 155
CVector::= 243	== 155
CVector::== 243	> 155
CVector::> 243	>= 156
CVector::>= 244	CVersion 150, 151
CVector::Allocate 232	GetBuild 151
CVector::AllocateSorted 232	GetMajor 151

GetMinor 152
GetRelease 152
GetStr 152
IsInRange 152
SetBuild 153
SetMajor 153
SetMinor 153
SetRelease 153
SetStr 154
CVersion::!= 154
CVersion::~CVersion 151
CVersion::< 154
CVersion::<= 155
CVersion::= 155
CVersion::== 155
CVersion::> 155
CVersion::>= 156
CVersion::CVersion 150, 151
CVersion::GetBuild 151
CVersion::GetMajor 151
CVersion::GetMinor 152
CVersion::GetRelease 152
CVersion::GetStr 152
CVersion::IsInRange 152
CVersion::SetBuild 153
CVersion::SetMajor 153
CVersion::SetMinor 153
CVersion::SetRelease 153
CVersion::SetStr 154
CVList 244
CVList template 244
!= 258
[] 259
~CVList 248
< 259
<= 259
= 260
== 260
> 260
>= 261
Allocate 248
AllocateSorted 249
Append 249
CVList 248
Erase 249, 250
EraseAll 250
EraseSorted 250
Find 250
FindPtrSorted 251
FindSorted 251
GetAt 251, 252
GetCapacity 252
GetEndIndex 252
GetFirstIndex 252
GetLastIndex 253
GetLockCapacity 253
GetMaxElementIndex 253
GetMinElementIndex 253
GetSize 254
Insert 254, 255
InsertSorted 255
IsEmpty 255
IsFull 255
LockCapacity 256
Merge 256
ReduceCapacity 256
ReserveCapacity 256
SetComparisonFunction 257
Sort 257
Split 257
Swap 258
UnlockCapacity 258
CVList::!= 258
CVList::[] 259
CVList::~CVList 248
CVList::< 259
CVList::<= 259
CVList::= 260
CVList::== 260
CVList::> 260
CVList::>= 261
CVList::Allocate 248
CVList::AllocateSorted 249
CVList::Append 249
CVList::CVList 248
CVList::Erase 249, 250

CVList::EraseAll 250	GetName 856
CVList::EraseSorted 250	GetNamespace 856
CVList::Find 250	GetNamespaceByPrefix 857
CVList::FindPtrSorted 251	GetNamespaceByUri 857
CVList::FindSorted 251	GetNamespacePrefix 857
CVList::GetAt 251, 252	GetNextSibling 858
CVList::GetCapacity 252	GetNumAttributes 858
CVList::GetEndIndex 252	GetNumChildElement 858
CVList::GetFirstIndex 252	GetOpaque 858
CVList::GetLastIndex 253	GetParentElement 859
CVList::GetLockCapacity 253	GetPreviousSibling 859
CVList::GetMaxElementIndex 253	GetValue 860
CVList::GetMinElementIndex 253	GetXmlDocument 860
CVList::GetSize 254	Serialize 860
CVList::Insert 254, 255	SetAttribute 860
CVList::InsertSorted 255	SetChildElement 861
CVList::IsEmpty 255	SetName 864
CVList::IsFull 255	SetNamespace 864
CVList::LockCapacity 256	SetOpaque 864
CVList::Merge 256	SetValue 865
CVList::ReduceCapacity 256	UpdateAttribute 865
CVList::ReserveCapacity 256	UpdateDeclaredNamespace 865
CVList::SetComparisonFunction 257	CXmlElement::AppendAttribute 843
CVList::Sort 257	CXmlElement::Copy 843
CVList::Split 257	CXmlElement::CreateChildElement 844, 845
CVList::Swap 258	CXmlElement::CreateElement 848, 849, 850
CVList::UnlockCapacity 258	CXmlElement::DeclareNamespace 851
CXmlElement 840	CXmlElement::Delete 852
CXmlElement class 840	CXmlElement::DeleteAttribute 852
AppendAttribute 843	CXmlElement::DeleteDeclaredNamespace 853
Copy 843	CXmlElement::ECreatePosition 866
CreateChildElement 844, 845	CXmlElement::ECREATEPOSITION enumeration 866
CreateElement 848, 849, 850	CXmlElement::ENsDeclarationBehavior 866
DeclareNamespace 851	CXmlElement::ENsDeclarationBehavior enumeration 866
Delete 852	CXmlElement::ENsDeclarationPosition 867
DeleteAttribute 852	CXmlElement::ENsDeclarationPosition enumeration 867
DeleteDeclaredNamespace 853	CXmlElement::FindChildElement 853
FindChildElement 853	CXmlElement::GetAttribute 854
GetAttribute 854	CXmlElement::GetChildElement 854, 855
GetChildElement 854, 855	CXmlElement::GetDeclaredNamespaces 855
GetDeclaredNamespaces 855	CXmlElement::GetFirstSibling 855
GetFirstSibling 855	CXmlElement::GetLastSibling 856
GetLastSibling 856	CXmlElement::GetName 856

CXmlElement::GetNamespace 856
CXmlElement::GetNamespaceByPrefix 857
CXmlElement::GetNamespaceByUri 857
CXmlElement::GetNamespacePrefix 857
CXmlElement::GetNextSibling 858
CXmlElement::GetNumAttributes 858
CXmlElement::GetNumChildElement 858
CXmlElement::GetOpaque 858
CXmlElement::GetParentElement 859
CXmlElement::GetPreviousSibling 859
CXmlElement::GetValue 860
CXmlElement::GetXmlDocument 860
CXmlElement::Serialize 860
CXmlElement::SetAttribute 860
CXmlElement::SetChildElement 861
CXmlElement::SetName 864
CXmlElement::SetNamespace 864
CXmlElement::SetOpaque 864
CXmlElement::SetValue 865
CXmlElement::SIIdentificationInfo 841
CXmlElement::SIIdentificationInfo struct 841
 m_pszName 841
 m_pszNamespace 841
 SIIdentificationInfo 842
CXmlElement::SIIdentificationInfo::m_pszName 841
CXmlElement::SIIdentificationInfo::m_pszNamespace 841
CXmlElement::SIIdentificationInfo::SIIdentificationInfo 842
CXmlElement::SNamespace 842
CXmlElement::SNamespace struct 842
 m_cCharacteristics 842
 m_pstNextNamespace 842
 m_pszPrefix 842
 m_pszUri 842
 SNamespace 843
CXmlElement::SNamespace::ENamespaceCharacteristics 843
CXmlElement::SNamespace::ENamespaceCharacteristics enumeration 843
CXmlElement::SNamespace::m_cCharacteristics 842
CXmlElement::SNamespace::m_pstNextNamespace 842
CXmlElement::SNamespace::m_pszPrefix 842
CXmlElement::SNamespace::m_pszUri 842
CXmlElement::SNamespace::SNamespace 843
CXmlElement::UpdateAttribute 865
CXmlElement::UpdateDeclaredNamespace 865
CXmlParser 867
CXmlParser class 867
 ~CXmlParser 868
 CXmlParser 868
 Parse 868
 SkipElementContent 869
CXmlParser::~CXmlParser 868
CXmlParser::CXmlParser 868
CXmlParser::Parse 868
CXmlParser::SkipElementContent 869
CXmlWriter 869
CXmlWriter class 869
 ~CXmlWriter 870
 CXmlWriter 870
 EmptyElement 871
 EndDocument 871
 EndElement 871
 EndElementAttribute 871
 EndEmptyElementAttribute 872
 Initialize 872
 StartDocument 872
 StartElement 873
 StartElementAttribute 873
 WriteAttributes 873
 WriteElement 874
CXmlWriter::~CXmlWriter 870
CXmlWriter::CXmlWriter 870
CXmlWriter::EEolMode 874
CXmlWriter::EEolMode enumeration 874
CXmlWriter::EIndentingMode 883
CXmlWriter::EIndentingMode enumeration 883
CXmlWriter::EmptyElement 871
CXmlWriter::EndDocument 871
CXmlWriter::EndElement 871
CXmlWriter::EndElementAttribute 871
CXmlWriter::EndEmptyElementAttribute 872
CXmlWriter::Initialize 872
CXmlWriter::StartDocument 872
CXmlWriter::StartElement 873
CXmlWriter::StartElementAttribute 873
CXmlWriter::WriteAttributes 873
CXmlWriter::WriteElement 874

D

Deprecated Macros 62
 Deprecated parameters qualifier 62
 Diffie-Hellman crypto algorithm configuration macros 282

E

eACQUIRED enumeration member 504
 eACTION_DECRYPT enumeration member 410
 eACTION_DEFAULT enumeration member 410
 eACTION_ENCRYPT enumeration member 410
 eACTIVE enumeration member 506
 eALGORITHM_AES enumeration member 410
 eALGORITHM_BASE64 enumeration member 410
 eALGORITHM_DEFAULT enumeration member 411
 eALGORITHM_MD5 enumeration member 411
 eALGORITHM_SHA1 enumeration member 411
 eALGORITHM_SHA2_224 enumeration member 411
 eALGORITHM_SHA2_256 enumeration member 411
 eALGORITHM_SHA2_384 enumeration member 411
 eALGORITHM_SHA2_512 enumeration member 411
 eALL_ACQUIRED enumeration member 504
 eALL_FREE enumeration member 504
 eAPPEND enumeration member 504
 eBINARY enumeration member 504
 ECom 412
 eCOMPACT enumeration member 883
 eCR enumeration member 874
 eCRC16 enumeration member 410
 eCRC32 enumeration member 410
 eCRC32C enumeration member 410
 eCREATE enumeration member 504
 eCRLF enumeration member 874
 eDATAGRAM enumeration member 652
 eDELETED enumeration member 506
 eEN_CMEMORYALLOCATOR_OUT_OF_MEMORY enumeration member 11
 eEN_CMEMORYALLOCATOR_TRACKING enumeration member 11
 eEN_FQDN_NOT_RESOLVED enumeration member 11
 eEN_LAST_EVENT enumeration member 11
 EEventNotifier 11
 EEventNotifier enumeration 11
 eFORCE enumeration member 651
 eFREE enumeration member 504
 eFRIDAY enumeration member 838
 eGRACEFUL enumeration member 651
 eHIGH enumeration member 506
 eHIGHEST enumeration member 506
 eINDENT enumeration member 883
 eINET enumeration member 651
 eINET_ANY enumeration member 651
 eINET_BROADCAST enumeration member 651
 eINET_LOOPBACK enumeration member 651
 eINET_NONE enumeration member 651
 eINET6 enumeration member 651
 eINET6_ANY enumeration member 651
 eINET6_LOOPBACK enumeration member 651
 eINTERNET_V4 enumeration member 651
 eINTERNET_V6 enumeration member 651
 eLF enumeration member 874
 eLINUX_24 enumeration member 505
 eLINUX_26 enumeration member 505
 eLOW enumeration member 506
 eLOWEST enumeration member 506
 eMEMORY_BLOCK_POOLED enumeration member 505
 eMEMORY_BLOCK_TEMPORARILY_UNTRACKED_FLAG enumeration member 505
 eMODE_CBC enumeration member 410
 eMODE_CFB enumeration member 410
 eMODE_CTR enumeration member 410
 eMODE_DEFAULT enumeration member 410
 eMODE_ECB enumeration member 410
 eMODE_OFB enumeration member 410
 eMONDAY enumeration member 838
 EMxBase 11
 EMxBase enumeration 11
 EMxPackageId 12
 EMxPackageId enumeration 12
 EMxResultSharedFailCriticalCodeId 13
 EMxResultSharedFailCriticalCodeId enumeration 13
 EMxResultSharedFailErrorCodeId 13
 EMxResultSharedFailErrorCodeId enumeration 13
 EMxResultSharedSuccessInfoCodeId 14
 EMxResultSharedSuccessInfoCodeId enumeration 14
 EMxResultSharedSuccessWarningCodeId 15

EMxResultSharedSuccessWarningCodeId enumeration 15
 EMxTraceField 15
 EMxTraceField enumeration 15
 EMxTraceLevel 16
 EMxTraceLevel enumeration 16
 eNO_VERSION_FOUND enumeration member 505
 eNONE enumeration member 874
 eNORMAL enumeration member 506
 eNSDECLARE_FORCE enumeration member 866
 eNSDECLARE_NONE enumeration member 866
 eNSDECLARE_OPTIMIZE enumeration member 866
 eNSDECLARE_OPTIMIZE_FAIL_ON_CONFLICT enumeration member 866
 eNSPOS_AS_COPY enumeration member 867
 eNSPOS_NEW_ELEMENT enumeration member 867
 eNSPOS_PARENT_ELEMENT enumeration member 867
 eNSPOS_ROOT_ELEMENT enumeration member 867
 eNSPOS_THIS_ELEMENT enumeration member 867
 Enumerations 10, 409, 463, 504, 650, 789, 815, 838, 883
 ePERIODICITY_NOT_PERIODIC enumeration member 789
 ePERIODICITY_REAJUST_WITH_CURRENT_TIME enumeration member 789
 ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_CYCLE_LOST enumeration member 789
 ePERIODICITY_REAJUST_WITH_PREVIOUS_TIME_NO_CYCLE_LOST enumeration member 789
 ePOS_FIRST_CHILD enumeration member 866
 ePOS_FIRST_SIBLING enumeration member 866
 ePOS_LAST_CHILD enumeration member 866
 ePOS_LAST_SIBLING enumeration member 866
 ePOS_NEXT_SIBLING enumeration member 866
 ePOS_PREVIOUS_SIBLING enumeration member 866
 eREAD_ONLY enumeration member 504
 eREAD_WRITE enumeration member 504
 eSATURDAY enumeration member 838
 eSEEK_CUR enumeration member 505
 eSEEK_END enumeration member 505
 eSEEK_SET enumeration member 505
 eSEQPACKET enumeration member 652
 eSOLARIS_10 enumeration member 505
 eSOLARIS_8 enumeration member 505
 eSTANDARD enumeration member 409
 eSTARTING enumeration member 506
 eSTREAM enumeration member 652
 eSUNDAY enumeration member 838
 eTERMINATED enumeration member 506
 eTEXT enumeration member 504
 eTHURSDAY enumeration member 838
 eTLS_VERSION_SSL_3_0 enumeration member 816
 eTLS_VERSION_TLS_1_0 enumeration member 816
 ETlsVersion 816
 ETlsVersion enumeration 816
 eTRUNCATE enumeration member 504
 eTUESDAY enumeration member 838
 eTYPE_PRINCIPAL enumeration member 464
 eTYPE_SRV_HST enumeration member 464
 eTYPE_SRV_INST enumeration member 464
 eTYPE_UNKNOWN enumeration member 464
 eUNKNOWN enumeration member 651
 eUNSUPPORTED_OS enumeration member 505
 eURI_MUST_BE_DELETED enumeration member 843
 eURL enumeration member 409
 eUSER_R enumeration member 505
 eUSER_RX enumeration member 505
 eUSER_W enumeration member 505
 eUSER_X enumeration member 505
 eWEDNESDAY enumeration member 838
 eWINDOWS_2003_PLUS enumeration member 505
 eWINDOWS_2K enumeration member 505
 eWINDOWS_NT enumeration member 505
 eWINDOWS_VISTA enumeration member 505
 eWINDOWS_XP enumeration member 505
 eWRITE_ONLY enumeration member 504

F

friend class CAsyncResolver 753, 756, 760
 friend class CHostFile 749
 friend class CResolverCache 749
 friend class Foo 391
 Functions 17, 420, 506, 652

G

g_stFramework 7
 g_stFramework and descendant tracing nodes 7
 g_stFrameworkBasic 7

g_stFrameworkBasic variable 7
g_stFrameworkCap 7
g_stFrameworkCap variable 7
g_stFrameworkCrypto 7
g_stFrameworkCrypto variable 7
g_stFrameworkECom 7
g_stFrameworkECom variable 7
g_stFrameworkEComCEComUnknown 7
g_stFrameworkEComCEComUnknown variable 7
g_stFrameworkKerberos 7
g_stFrameworkKerberos variable 7
g_stFrameworkKernel 7
g_stFrameworkKernel variable 7
g_stFrameworkKernelCFile 7
g_stFrameworkKernelCFile variable 7
g_stFrameworkMocanaSs 7
g_stFrameworkMocanaSs variable 7
g_stFrameworkNetwork 7
g_stFrameworkNetwork variable 7
g_stFrameworkNetworkCAsyncSocketFactory 7
g_stFrameworkNetworkCAsyncSocketFactory variable 7
g_stFrameworkNetworkCAsyncTcpServerSocket 7
g_stFrameworkNetworkCAsyncTcpServerSocket variable 7
g_stFrameworkNetworkCAsyncTcpSocket 7
g_stFrameworkNetworkCAsyncTcpSocket variable 7
g_stFrameworkNetworkCAsyncUdpSocket 7
g_stFrameworkNetworkCAsyncUdpSocket variable 7
g_stFrameworkNetworkCPollRequestStatus 7
g_stFrameworkNetworkCPollRequestStatus variable 7
g_stFrameworkNetworkCPollRequestStatusPoll 7
g_stFrameworkNetworkCPollRequestStatusPoll variable 7
g_stFrameworkNetworkCPollSocket 7
g_stFrameworkNetworkCPollSocket variable 7
g_stFrameworkNetworkCPollSocketPoll 7
g_stFrameworkNetworkCPollSocketPoll variable 7
g_stFrameworkNetworkCTcpServerSocket 7
g_stFrameworkNetworkCTcpServerSocket variable 7
g_stFrameworkNetworkCTcpSocket 7
g_stFrameworkNetworkCTcpSocket variable 7
g_stFrameworkNetworkCTcpSocketSendRecv 7
g_stFrameworkNetworkCTcpSocketSendRecv variable 7
g_stFrameworkNetworkCUdpSocket 7
g_stFrameworkNetworkCUdpSocket variable 7
g_stFrameworkNetworkNetworkSocketErrors 7
g_stFrameworkNetworkSocketErrors variable 7
g_stFrameworkPki 7
g_stFrameworkPki variable 7
g_stFrameworkRegExp 7
g_stFrameworkRegExp variable 7
g_stFrameworkResolver 7
g_stFrameworkResolver variable 7
g_stFrameworkServicingThread 7
g_stFrameworkServicingThread variable 7
g_stFrameworkServicingThreadCEventDriven 7
g_stFrameworkServicingThreadCEventDriven variable 7
g_stFrameworkServicingThreadCServicingThread 7
g_stFrameworkServicingThreadCServicingThread variable 7
g_stFrameworkServicingThreadCServicingThreadActivate 7
g_stFrameworkServicingThreadCServicingThreadActivate variable 7
g_stFrameworkServicingThreadCServicingThreadMessageService 7
g_stFrameworkServicingThreadCServicingThreadMessageService variable 7
g_stFrameworkServicingThreadCServicingThreadSocketService 7
g_stFrameworkServicingThreadCServicingThreadSocketService variable 7
g_stFrameworkServicingThreadCServicingThreadTimerService 7
g_stFrameworkServicingThreadCServicingThreadTimerService variable 7
g_stFrameworkTime 7
g_stFrameworkTime variable 7
g_stFrameworkTls 7
g_stFrameworkTls variable 7
g_stFrameworkTlsCAsyncTlsServerSocket 7
g_stFrameworkTlsCAsyncTlsServerSocket variable 7
g_stFrameworkTlsCAsyncTlsServerSocketBase 7
g_stFrameworkTlsCAsyncTlsServerSocketBase variable 7
g_stFrameworkTlsCAsyncTlsSocket 7
g_stFrameworkTlsCAsyncTlsSocket variable 7
g_stFrameworkTlsCAsyncTlsSocketBase 7
g_stFrameworkTlsCAsyncTlsSocketBase variable 7
g_stFrameworkTlsCAsyncTlsSocketFactoryCreationMgr 7
g_stFrameworkTlsCAsyncTlsSocketFactoryCreationMgr variable 7
g_stFrameworkTlsCTlsContext 7
g_stFrameworkTlsCTlsContext variable 7
g_stFrameworkTlsCTlsSession 7
g_stFrameworkTlsCTlsSession variable 7

g_stFrameworkTIsCTIsSessionMocanaSs 7	GetHostByAddr function 653
g_stFrameworkTIsCTIsSessionMocanaSs variable 7	GetHostByName 654
g_stFrameworkTIsCTIsSessionOpenSsl 7	GetHostByName function 654
g_stFrameworkTIsCTIsSessionOpenSsl variable 7	GetLocalHostName 654
g_stFrameworkTIsCTIsSocketSendRecv 7	GetLocalHostName function 654
g_stFrameworkTIsCTIsSocketSendRecv variable 7	GetLocalIPAddr 655
g_stFrameworkXml 7	GetLocalIPAddr function 655
g_stFrameworkXml variable 7	GetLocalIPv4Addr 655
g_stFrameworkXmlDocument 7	GetLocalIPv4Addr function 655
g_stFrameworkXmlDocument variable 7	GetLocalIPv6Addr 655
g_stFrameworkXmlDocumentEcom 7	GetLocalIPv6Addr function 655
g_stFrameworkXmlDocumentEcom variable 7	GetNaptrRecord 655
g_stFrameworkXmlElement 7	GetNaptrRecord function 655
g_stFrameworkXmlElement variable 7	GetSocketErrorId 656
g_stFrameworkXmlGenericWriter 7	GetSocketErrorId function 656
g_stFrameworkXmlGenericWriter variable 7	GetSockOptError 599
g_stFrameworkXmlGenericWriterEcom 7	GetSockOptError function 599
g_stFrameworkXmlGenericWriterEcom variable 7	GetSrvRecord 656, 657
g_stFrameworkXmlParserExpat 7	GetSrvRecord function 656, 657
g_stFrameworkXmlParserExpat variable 7	GO 39
g_stFrameworkXmlParserExpatEcom 7	GO macro 39
g_stFrameworkXmlParserExpatEcom variable 7	
g_stTraceRoot 7	
g_stTraceRoot tracing node 7	
g_szEMPTY_STRING 94	
g_szEMPTY_STRING variable 94	
g_szNULL 94	
g_szNULL variable 94	
g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEA KED_MEMORY_BLOCKS 794	
g_uFRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEA KED_MEMORY_BLOCKS variable 794	
g_uMEMORY_BLOCK_OVERHEAD_SIZE 521	
g_uMEMORY_BLOCK_OVERHEAD_SIZE variable 521	
g_uTHREAD_NAME_MAX_SIZE 522	
g_uTHREAD_NAME_MAX_SIZE variable 522	
General Tracing Configuration 2	
GetAllLocalIpAddresses 653	
GetAllLocalIpAddresses function 653	
GetEnumUri 653	
GetEnumUri function 653	
GetHostByAddr 653	
	GetHostByName 654
	GetHostByName function 654
	GetLocalHostName 654
	GetLocalHostName function 654
	GetLocalIPAddr 655
	GetLocalIPAddr function 655
	GetLocalIPv4Addr 655
	GetLocalIPv4Addr function 655
	GetLocalIPv6Addr 655
	GetLocalIPv6Addr function 655
	GetNaptrRecord 655
	GetNaptrRecord function 655
	GetSocketErrorId 656
	GetSocketErrorId function 656
	GetSockOptError 599
	GetSockOptError function 599
	GetSrvRecord 656, 657
	GetSrvRecord function 656, 657
	GO 39
	GO macro 39
	H
	Hexadecimal tracing macros 54
	HexToAscii 18
	HexToAscii function 18
	I
	IActivationService 775
	IActivationService class 775
	Activate 776
	IsCurrentExecutionContext 777
	IActivationService::Activate 776
	IActivationService::IsCurrentExecutionContext 777
	IAlocator 156
	IAlocator class 156
	Allocate 157
	Release 157
	IAlocator::Allocate 157
	IAlocator::Release 157
	IAlocator::SBlock 157
	IAlocator::SBlock struct 157

IAsyncClientSocket 599	IAsyncServerSocket::BindA 608
IAsyncClientSocket class 599	IAsyncServerSocket::Listen 609
BindA 600	IAsyncServerSocket::QueryAcceptedOptionsIf 609
ConnectA 600	IAsyncServerSocket::SetAsyncServerSocketMgr 610
SetAsyncClientSocketMgr 600	IAsyncServerSocketMgr 610
IAsyncClientSocket::BindA 600	IAsyncServerSocketMgr class 610
IAsyncClientSocket::ConnectA 600	EvAsyncServerSocketMgrBound 611
IAsyncClientSocket::SetAsyncClientSocketMgr 600	EvAsyncServerSocketMgrConnectionAccepted 611
IAsyncClientSocketMgr 601	EvAsyncServerSocketMgrConnectionFailed 612
IAsyncClientSocketMgr class 601	EvAsyncServerSocketMgrConnectionRequested 612
EvAsyncClientSocketMgrBound 601	IAsyncServerSocketMgr::EvAsyncServerSocketMgrBound 611
EvAsyncClientSocketMgrConnected 602	IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionAccepted 611
IAsyncClientSocketMgr::EvAsyncClientSocketMgrBound 601	IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionFailed 612
IAsyncClientSocketMgr::EvAsyncClientSocketMgrConnected 602	IAsyncServerSocketMgr::EvAsyncServerSocketMgrConnectionRequested 612
IAsyncloSocket 602	IAsyncSocket 612
IAsyncloSocket class 602	IAsyncSocket class 612
GetPeerAddress 603	Activate 613
Recv 603, 604	CloseA 614
Send 604, 605	EraseAllUserInfo 614
SetAsyncloSocketMgr 605	EraseUserInfo 615
IAsyncloSocket::GetPeerAddress 603	GetHandle 615
IAsyncloSocket::Recv 603, 604	GetLocalAddress 615
IAsyncloSocket::Send 604, 605	GetLocalInterfaceAddress 616
IAsyncloSocket::SetAsyncloSocketMgr 605	GetOpaque 616
IAsyncloSocketMgr 605	GetServicingThreadIcomUnknown 616
IAsyncloSocketMgr class 605	GetSocketType 617
EvAsyncloSocketMgrReadyToRecv 606	GetUserInfo 618
EvAsyncloSocketMgrReadyToSend 606	InsertUserInfo 618
IAsyncloSocketMgr::EvAsyncloSocketMgrReadyToRecv 606	SetAsyncSocketMgr 619
IAsyncloSocketMgr::EvAsyncloSocketMgrReadyToSend 606	SetOpaque 619
IAsyncResolverUser 760	SetSocketType 619
IAsyncResolverUser class 760	IAsyncSocket::Activate 613
EvAsyncResolverUserResponseReceived 760, 761	IAsyncSocket::CloseA 614
IAsyncResolverUser::EvAsyncResolverUserResponseReceived 760, 761	IAsyncSocket::EraseAllUserInfo 614
IAsyncServerSocket 607	IAsyncSocket::EraseUserInfo 615
IAsyncServerSocket class 607	IAsyncSocket::GetHandle 615
AcceptA 607	IAsyncSocket::GetLocalAddress 615
BindA 608	IAsyncSocket::GetLocalInterfaceAddress 616
Listen 609	IAsyncSocket::GetOpaque 616
QueryAcceptedOptionsIf 609	
SetAsyncServerSocketMgr 610	
IAsyncServerSocket::AcceptA 607	

IAsyncSocket::GetServicingThreadIcomUnknown	616	IAsyncSocketTcpOptions::SetNagle	628
IAsyncSocket::GetSocketType	617	IAsyncSocketUdpOptions	629
IAsyncSocket::GetUserInfo	618	IAsyncSocketUdpOptions class	629
IAsyncSocket::InsertUserInfo	618	SetBroadcast	629
IAsyncSocket::SetAsyncSocketMgr	619	IAsyncSocketUdpOptions::SetBroadcast	629
IAsyncSocket::SetOpaque	619	IAsyncSocketWindowsGqosOptions	630
IAsyncSocket::SetSocketType	619	IAsyncSocketWindowsGqosOptions class	630
IAsyncSocketBufferSizeOptions	620	SetWindowsReceivingFlowspec	630
IAsyncSocketBufferSizeOptions class	620	SetWindowsSendingFlowspec	630
SetReceiveBufferSize	621	IAsyncSocketWindowsGqosOptions::SetWindowsReceivingFlowspec	
SetTransmitBufferSize	621	c	
IAsyncSocketBufferSizeOptions::SetReceiveBufferSize	621	630	
IAsyncSocketBufferSizeOptions::SetTransmitBufferSize	621	IAsyncSocketWindowsGqosOptions::SetWindowsSendingFlowspec	
IAsyncSocketFactoryConfigurationMgr	621	630	
IAsyncSocketFactoryConfigurationMgr class	621	IAsyncTlsOpenSsl	806
EvConfigurationRequested	622	IAsyncTlsOpenSsl class	806
IAsyncSocketFactoryConfigurationMgr::EvConfigurationRequested	622	GetSsl	807
IAsyncSocketFactoryCreationMgr	622	IAsyncTlsOpenSsl::GetSsl	807
IAsyncSocketFactoryCreationMgr class	622	IAsyncTlsRenegotiation	807
EvCreationRequested	623	IAsyncTlsRenegotiation class	807
IAsyncSocketFactoryCreationMgr::EvCreationRequested	623	EnableAllRenegotiationNotifications	808
IAsyncSocketMgr	624	RenegotiateA	808
IAsyncSocketMgr class	624	SetAsyncTlsRenegotiationMgr	808
EvAsyncSocketMgrClosed	625	SetAutoRenegotiationThresholdInByte	809
EvAsyncSocketMgrClosedByPeer	625	SetAutoRenegotiationThresholdInTimeMs	809
EvAsyncSocketMgrErrorDetected	625	SetAutoRenegotiationTimeoutMs	809
IAsyncSocketMgr::EvAsyncSocketMgrClosed	625	IAsyncTlsRenegotiation::EnableAllRenegotiationNotifications	808
IAsyncSocketMgr::EvAsyncSocketMgrClosedByPeer	625	IAsyncTlsRenegotiation::RenegotiateA	808
IAsyncSocketMgr::EvAsyncSocketMgrErrorDetected	625	IAsyncTlsRenegotiation::SetAsyncTlsRenegotiationMgr	808
IAsyncSocketQualityOfServiceOptions	625	IAsyncTlsRenegotiation::SetAutoRenegotiationThresholdInByte	809
IAsyncSocketQualityOfServiceOptions class	625	IAsyncTlsRenegotiation::SetAutoRenegotiationThresholdInTimeMs	809
Set8021QUserPriority	626	IAsyncTlsRenegotiation::SetAutoRenegotiationTimeoutMs	809
SetTos	627	IAsyncTlsRenegotiationMgr	810
IAsyncSocketQualityOfServiceOptions::Set8021QUserPriority	626	IAsyncTlsRenegotiationMgr class	810
IAsyncSocketQualityOfServiceOptions::SetTos	627	EvAsyncTlsRenegotiationMgrRenegotiated	810
IAsyncSocketTcpOptions	627	IAsyncTlsRenegotiationMgr::EvAsyncTlsRenegotiationMgrRenegotiated	810
IAsyncSocketTcpOptions class	627	IAsyncTlsServerSocket	811
SetConnectTimeoutMs	628	IAsyncTlsServerSocket class	811
SetKeepAlive	628	GetAcceptedTlsContext	811
SetNagle	628	SetAcceptedTlsContext	812
IAsyncSocketTcpOptions::SetConnectTimeoutMs	628	IAsyncTlsServerSocket::GetAcceptedTlsContext	811
IAsyncSocketTcpOptions::SetKeepAlive	628	IAsyncTlsServerSocket::SetAcceptedTlsContext	812

IAsyncTlsSocket 812	IClientSocket::Connect 637
IAsyncTlsSocket class 812	IEComUnknown 416
GetTlsContext 813	IEComUnknown class 416
SetAsyncTlsSocketMgr 813	AddIfRef 418
SetTlsContext 813	QueryIf 418, 419
SetTlsSession 814	ReleaseIfRef 420
TlsHandshakingApprovalCompletedA 814	IEComUnknown::AddIfRef 418
IAsyncTlsSocket::GetTlsContext 813	IEComUnknown::QueryIf 418, 419
IAsyncTlsSocket::SetAsyncTlsSocketMgr 813	IEComUnknown::ReleaseIfRef 420
IAsyncTlsSocket::SetTlsContext 813	IExternalTimeSource 837
IAsyncTlsSocket::SetTlsSession 814	IExternalTimeSource class 837
IAsyncTlsSocket::TlsHandshakingApprovalCompletedA 814	GetTimeS 838
IAsyncTlsSocketMgr 814	IExternalTimeSource::GetTimeS 838
IAsyncTlsSocketMgr class 814	IIoSocket 637
EvAsyncTlsSocketMgrNewTlsSession 815	IIoSocket class 637
EvAsyncTlsSocketMgrTlsHandshakingCompletedAwaitingAppr	GetPeerAddress 638
oval 815	Recv 638, 639
IAsyncTlsSocketMgr::EvAsyncTlsSocketMgrNewTlsSession 815	RecvFrom 639, 640
IAsyncTlsSocketMgr::EvAsyncTlsSocketMgrTlsHandshakingCompl	Send 640, 641
etedAwaitingApproval 815	SendTo 641, 642
IIoSocket 631	IIoSocket::GetPeerAddress 638
IIoSocket class 631	IIoSocket::Recv 638, 639
RecvFrom 632	IIoSocket::RecvFrom 639, 640
SendTo 633	IIoSocket::Send 640, 641
SetAsyncUnconnectedIoSocketMgr 634	IIoSocket::SendTo 641, 642
IIoSocket::RecvFrom 632	IKerberosTicketRequestMgr 462
IIoSocket::SendTo 633	IKerberosTicketRequestMgr class 462
IIoSocket::SetAsyncUnconnectedIoSocketMgr 634	EvKerberosTicketRequestMgrGetTicketError 463
IIoSocketMgr 634	EvKerberosTicketRequestMgrGetTicketSuccess 463
IIoSocketMgr class 634	IKerberosTicketRequestMgr::EvKerberosTicketRequestMgrGetTick
EvAsyncUnconnectedIoSocketMgrReadyToRecv 635	etError 463
EvAsyncUnconnectedIoSocketMgrReadyToSend 635	EvKerberosTicketRequestMgrGetTicketSuccess 463
IIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToRecv 635	IMessageService 777
IIoSocketMgr::EvAsyncUnconnectedIoSocketMgrReadyToSend 635	IMessageService class 777
Bind 636	PostMessage 778
Connect 637	IMessageService::PostMessage 778
IClientSocket::Bind 636	IMessageServiceMgr 778
	IMessageServiceMgr class 778
	EvMessageServiceMgrAwaken 779
	IMessageServiceMgr::EvMessageServiceMgrAwaken 779
	IN 39

INOUT 39	IsEqualEComIID function 422
INOUT macro 39	IServerSocket 644
InstallGenericFaultHandler 19	IServerSocket class 644
InstallGenericFaultHandler function 19	Accept 645
int16_t 85	Bind 645
int16_t type 85	Listen 646
int32_t 85	IServerSocket::Accept 645
int32_t type 85	IServerSocket::Bind 645
int64_t 85	IServerSocket::Listen 646
int64_t type 85	ISocket 646
int8_t 85	ISocket class 646
Introduction 1	Close 647
IPassPhrase 408	GetAddressFamily 647
IPassPhrase class 408	GetLocalAddress 647
GetPassPhrase 409	GetProtocolFamily 648
IPassPhrase::GetPassPhrase 409	GetSocketType 648
IPolledRequestStatusMgr 642	Release 648
IPolledRequestStatusMgr class 642	Set8021QUserPriority 649
EvPolledRequestStatusMgrEventDetected 643	SetTos 649
IPolledRequestStatusMgr::EvPolledRequestStatusMgrEventDetected 643	ISocket::Close 647
IPolledSocketMgr 643	ISocket::ECloseBehavior 651
IPolledSocketMgr class 643	ISocket::ECloseBehavior enumeration 651
EvPolledSocketMgrEventDetected 644	ISocket::EProtocolFamily 651
IPolledSocketMgr::EvPolledSocketMgrEventDetected 644	ISocket::EProtocolFamily enumeration 651
IRquestStatusService 779	ISocket::ESocketType 652
IRquestStatusService class 779	ISocket::ESocketType enumeration 652
DisableCompletionDetection 780	ISocket::GetAddressFamily 647
EnableCompletionDetection 780	ISocket::GetLocalAddress 647
RegisterRequestStatus 781	ISocket::GetProtocolFamily 648
UnregisterRequestStatus 781	ISocket::GetSocketType 648
IRquestStatusService::DisableCompletionDetection 780	ISocket::Release 648
IRquestStatusService::EnableCompletionDetection 780	ISocket::Set8021QUserPriority 649
IRquestStatusService::RegisterRequestStatus 781	ISocket::SetTos 649
IRquestStatusService::UnregisterRequestStatus 781	ISocketService 782
IRquestStatusServiceMgr 781	ISocketService class 782
IRquestStatusServiceMgr class 781	DisableEventsDetection 783
EvRequestStatusServiceMgrAwaken 782	EnableEventsDetection 783
IRquestStatusServiceMgr::EvRequestStatusServiceMgrAwaken 782	RegisterSocket 784
IsEqualECom 422	UnregisterSocket 784
IsEqualECom function 422	ISocketService::DisableEventsDetection 783
IsEqualEComIID 422	ISocketService::EnableEventsDetection 783
	ISocketService::RegisterSocket 784
	ISocketService::UnregisterSocket 784

ISocketServiceMgr 784	IXmlDocument::GetPatchManager 877
ISocketServiceMgr class 784	IXmlDocument::GetRootElement 877
EvSocketServiceMgrAwaken 785	IXmlDocument::Parse 878
ISocketServiceMgr::EvSocketServiceMgrAwaken 785	IXmlDocument::Serialize 878
ITcpSocketOptionsConfigure 650	IXmlDocument::SetAllocator 879
ITcpSocketOptionsConfigure class 650	IXmlDocument::SetDictionary 879
ApplyOptions 650	IXmlDocument::SetDocumentManager 880
ITcpSocketOptionsConfigure::ApplyOptions 650	IXmlDocument::SetOpaque 880
ITimerService 785	IXmlDocument::SetPatchManager 880
ITimerService class 785	IXmlParserHandler 881
StartTimer 786	IXmlParserHandler class 881
StopAllTimers 787	EvXmlParserHandlerCharacterData 881
StopTimer 787	EvXmlParserHandlerComment 881
ITimerService::EPeriodicity 789	EvXmlParserHandlerDefault 882
ITimerService::EPeriodicity enumeration 789	EvXmlParserHandlerEndElement 882
ITimerService::StartTimer 786	EvXmlParserHandlerSkippedContent 882
ITimerService::StopAllTimers 787	EvXmlParserHandlerStartElement 882
ITimerService::StopTimer 787	IXmlParserHandler::EvXmlParserHandlerCharacterData 881
ITimerServiceMgr 788	IXmlParserHandler::EvXmlParserHandlerComment 881
ITimerServiceMgr class 788	IXmlParserHandler::EvXmlParserHandlerDefault 882
EvTimerServiceMgrAwaken 788	IXmlParserHandler::EvXmlParserHandlerEndElement 882
ITimerServiceMgr::EvTimerServiceMgrAwaken 788	IXmlParserHandler::EvXmlParserHandlerSkippedContent 882
IXmlDocument 874	IXmlParserHandler::EvXmlParserHandlerStartElement 882
IXmlDocument class 874	IXmlWriterOutputHandler 882
CreateRootElement 875	IXmlWriterOutputHandler class 882
DeleteRootElement 876	EvXmlWriterOutputHandlerWrite 883
GetDictionary 876	IXmlWriterOutputHandler::EvXmlWriterOutputHandlerWrite 883
GetDocumentManager 876	
GetOpaque 877	
GetPatchManager 877	
GetRootElement 877	
Parse 878	
Serialize 878	
SetAllocator 879	
SetDictionary 879	
SetDocumentManager 880	
SetOpaque 880	
SetPatchManager 880	
IXmlDocument::CreateRootElement 875	
IXmlDocument::DeleteRootElement 876	
IXmlDocument::GetDictionary 876	
IXmlDocument::GetDocumentManager 876	
IXmlDocument::GetOpaque 877	

K

Kerberos 426

Kernel 464

M

M5T Framework 2

M5T Framework tracing nodes 6

Macros 37, 261, 423, 509

Macros to get the base result code id 49

MD5 crypto algorithm configuration macros 282

MD5 MAC crypto algorithm configuration macros 283

MX_ALIGNMENT_OF 59

MX_ALIGNMENT_OF macro 59

MX_ASSERT 39

MX_ASSERT macro 39	MX_HI32 57
MX_ASSERT_EX 40	MX_HI32 macro 57
MX_ASSERT_EX macro 40	MX_HI8 44
MX_ASSERT_ONLY 40	MX_HI8 macro 44
MX_ASSERT_ONLY macro 40	MX_HTON64 44
MX_ASSERT_ONLY_RT 40	MX_HTON64 macro 44
MX_ASSERT_ONLY_RT macro 40	MX_HTONL 44
MX_ASSERT_PERROR 41	MX_HTONL macro 44
MX_ASSERT_PERROR macro 41	MX_HTONS 44
MX_ASSERT_PERROR_EX 41	MX_HTONS macro 44
MX_ASSERT_PERROR_EX macro 41	MX_INT32_TO_OPQ 60
MX_ASSERT_PERROR_RT 41	MX_INT32_TO_OPQ macro 60
MX_ASSERT_PERROR_RT macro 41	MX_ISDIGIT 62
MX_ASSERT_PERROR_RT_EX 42	MX_ISDIGIT macro 62
MX_ASSERT_PERROR_RT_EX macro 42	MX_LOW16 45
MX_ASSERT_RT 42	MX_LOW16 macro 45
MX_ASSERT_RT macro 42	MX_LOW32 45
MX_ASSERT_RT_EX 42	MX_LOW32 macro 45
MX_ASSERT_RT_EX macro 42	MX_LOW8 45
MX_DECLARE_DELEGATING_IECOMUNKNOWN 423	MX_LOW8 macro 45
MX_DECLARE_DELEGATING_IECOMUNKNOWN macro 423	MX_MAKE_NULL_EMPTY_STRING 58
MX_DECLARE_ECOM_CLSID 424	MX_MAKE_NULL_EMPTY_STRING macro 58
MX_DECLARE_ECOM_CLSID macro 424	MX_MAKE_STRING_NULL_SAFE 58
MX_DECLARE_ECOM_GETIID 424	MX_MAKE_STRING_NULL_SAFE macro 58
MX_DECLARE_ECOM_GETIID macro 424	MX_MAKEUINT16 46
MX_DECLARE_ECOM_IID 424	MX_MAKEUINT16 macro 46
MX_DECLARE_ECOM_IID macro 424	MX_MAKEUINT32 46
MX_DECLARE_IECOMUNKNOWN 425	MX_MAKEUINT32 macro 46
MX_DECLARE_IECOMUNKNOWN macro 425	MX_MAKEUINT64 57
MX_DEFINEINT64 43	MX_MAKEUINT64 macro 57
MX_DEFINEINT64 macro 43	MX_MAX 46
MX_DEFINEUINT64 43	MX_MAX macro 46
MX_DEFINEUINT64 macro 43	MX_MIN 46
MX_DELETE 509	MX_MIN macro 46
MX_DELETE macro 509	MX_NAMESPACE 47
MX_DELETE_ARRAY 510	MX_NAMESPACE macro 47
MX_DELETE_ARRAY macro 510	MX_NAMESPACE_END 47
MX_ERRNO_GET 58	MX_NAMESPACE_END macro 47
MX_ERRNO_GET macro 58	MX_NAMESPACE_START 47
MX_ERRNO_SET 57	MX_NAMESPACE_START macro 47
MX_ERRNO_SET macro 57	MX_NAMESPACE_USE 48
MX_HI16 43	MX_NAMESPACE_USE macro 48
MX_HI16 macro 43	MX_NEW 510

MX_NEW macro	510	MX_RGET_PKG_BASE_FAIL_CRITICAL_CODE_ID	49
MX_NEW_ARRAY	510	MX_RGET_PKG_BASE_FAIL_CRITICAL_CODE_ID macro	49
MX_NEW_ARRAY macro	510	MX_RGET_PKG_BASE_FAIL_ERROR_CODE_ID	49
MX_NTOH64	48	MX_RGET_PKG_BASE_FAIL_ERROR_CODE_ID macro	49
MX_NTOH64 macro	48	MX_RGET_PKG_BASE_SUCCESS_WARNING_CODE_ID	49
MX_NTOHL	48	MX_RGET_WORST_OF	50
MX_NTOHL macro	48	MX_RGET_WORST_OF macro	50
MX_NTOHS	48	MX_RIS_F	50
MX_NTOHS macro	48	MX_RIS_F macro	50
MX_OPQ_TO_INT32	60	MX_RIS_FC	50
MX_OPQ_TO_INT32 macro	60	MX_RIS_FC macro	50
MX_OPQ_TO_UINT32	61	MX_RIS_FE	50
MX_OPQ_TO_UINT32 macro	61	MX_RIS_FE macro	50
MX_OPQ_TO_VOIDPTR	59	MX_RIS_S	50
MX_OPQ_TO_VOIDPTR macro	59	MX_RIS_SI	50
MX_PACK MACROS	55	MX_RIS_SI macro	50
MX_PACK_STRUCT	55	MX_RIS_SW	50
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_BEGIN	49	MX_SIZEOFARRAY	51
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_BEGIN macro	49	MX_SIZEOFARRAY macro	51
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_END	49	MX_SWAPBYTES16	51
MX_R_PKG_FAIL_CRITICAL_MSG_TBL_END macro	49	MX_SWAPBYTES16 macro	51
MX_R_PKG_FAIL_ERROR_MSG_TBL_BEGIN	49	MX_SWAPBYTES32	52
MX_R_PKG_FAIL_ERROR_MSG_TBL_BEGIN macro	49	MX_SWAPBYTES32 macro	52
MX_R_PKG_FAIL_ERROR_MSG_TBL_END	49	MX_SWAPBYTES64	57
MX_R_PKG_FAIL_ERROR_MSG_TBL_END macro	49	MX_SWAPBYTES64 macro	57
MX_R_PKG_SUCCESS_INFO_MSG_TBL_BEGIN	49	MX_TRACE	52
MX_R_PKG_SUCCESS_INFO_MSG_TBL_END	49	MX_TRACE_ALL_CALLSTACKS	53
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_BEGIN	49	MX_TRACE_ALL_CALLSTACKS macro	53
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_BEGIN macro	49	MX_TRACE_CALLSTACK	53
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_END	49	MX_TRACE_CALLSTACK_COREDUMP	53
MX_R_PKG_SUCCESS_WARNING_MSG_TBL_END macro	49	MX_TRACE_CALLSTACK_COREDUMP macro	53
MX_REMOVE_UNUSED_FUNCTION_PARAM_WARNING	58	MX_TRACE_HEX	54
MX_REMOVE_UNUSED_FUNCTION_PARAM_WARNING macro	58	MX_TRACE_NEW_BUFFER	313
MX_RGET_CID	50	MX_TRACE_NEW_BUFFER macro	313
MX_RGET_CID macro	50	MX_TRACE_RELEASE_BUFFER	313
MX_RGET_LEV	50	MX_TRACE_RELEASE_BUFFER macro	313
MX_RGET_LEV macro	50	MX_TRACE0	52
MX_RGET_MSG_STR	50	MX_TRACE0 macro	52
MX_RGET_MSG_STR macro	50	MX_TRACE0_HEX	54
MX_RGET_PID	50	MX_TRACE0_HEX macro	54
MX_RGET_PID macro	50	MX_TRACE0_IS_ENABLED	54
		MX_TRACE1	52

MX_TRACE1 macro 52
MX_TRACE1_HEX 54
MX_TRACE1_HEX macro 54
MX_TRACE1_IS_ENABLED 54
MX_TRACE1_IS_ENABLED macro 54
MX_TRACE2 52
MX_TRACE2 macro 52
MX_TRACE2_HEX 54
MX_TRACE2_HEX macro 54
MX_TRACE2_IS_ENABLED 54
MX_TRACE2_IS_ENABLED macro 54
MX_TRACE3 52
MX_TRACE3 macro 52
MX_TRACE3_HEX 54
MX_TRACE3_HEX macro 54
MX_TRACE3_IS_ENABLED 54
MX_TRACE3_IS_ENABLED macro 54
MX_TRACE4 52
MX_TRACE4 macro 52
MX_TRACE4_HEX 54
MX_TRACE4_HEX macro 54
MX_TRACE4_IS_ENABLED 54
MX_TRACE4_IS_ENABLED macro 54
MX_TRACE5 52
MX_TRACE5 macro 52
MX_TRACE5_HEX 54
MX_TRACE5_HEX macro 54
MX_TRACE5_IS_ENABLED 54
MX_TRACE5_IS_ENABLED macro 54
MX_TRACE6 52
MX_TRACE6 macro 52
MX_TRACE6_HEX 54
MX_TRACE6_HEX macro 54
MX_TRACE6_IS_ENABLED 54
MX_TRACE6_IS_ENABLED macro 54
MX_TRACE7 52
MX_TRACE7 macro 52
MX_TRACE7_HEX 54
MX_TRACE7_HEX macro 54
MX_TRACE7_IS_ENABLED 54
MX_TRACE7_IS_ENABLED macro 54
MX_TRACE8 52
MX_TRACE8 macro 52
MX_TRACE8_HEX 54
MX_TRACE8_HEX macro 54
MX_TRACE8_IS_ENABLED 54
MX_TRACE8_IS_ENABLED macro 54
MX_TRACE9 52
MX_TRACE9 macro 52
MX_TRACE9_HEX 54
MX_TRACE9_HEX macro 54
MX_TRACE9_IS_ENABLED 54
MX_TRACE9_IS_ENABLED macro 54
MX_UINT32_TO_OPQ 61
MX_UINT32_TO_OPQ macro 61
MX_VOIDPTR_TO_OPQ 59
MX_VOIDPTR_TO_OPQ macro 59
MxAssertSetNewCallStackTraceHandler 19
MxAssertSetNewCallStackTraceHandler function 19
MxAssertSetNewFailHandler 19
MxAssertSetNewFailHandler function 19
MxAssertSetNewFinalBehaviorHandler 19
MxAssertSetNewFinalBehaviorHandler function 19
MxAssertSetNewTraceHandler 20
MxAssertSetNewTraceHandler function 20
MxChmod 507
MxChmod function 507
MxCloseSocket 657
MxCloseSocket function 657
MXD_64BITS_CUSTOM_TYPE 268
MXD_64BITS_CUSTOM_TYPE macro 268
MXD_64BITS_SUPPORT_DISABLE 268
MXD_64BITS_SUPPORT_DISABLE macro 268
MXD_ALIGNED_ACCESS_REQUIRED 269
MXD_ALIGNED_ACCESS_REQUIRED macro 269
MXD_ARCH_AMD64 269
MXD_ARCH_AMD64 macro 269
MXD_ARCH_ARM 269
MXD_ARCH_ARM macro 269
MXD_ARCH_IX86 269
MXD_ARCH_IX86 macro 269
MXD_ARCH_MIPS 270
MXD_ARCH_MIPS macro 270
MXD_ARCH_NIOS2 270
MXD_ARCH_NIOS2 macro 270
MXD_ARCH_PPC 270

MXD_ARCH_PPC macro	270	MXD_CAP_ENABLE_SUPPORT	277
MXD_ASSERT_CALL_STACK_TRACE_DISABLE	270	MXD_CAP_ENABLE_SUPPORT macro	277
MXD_ASSERT_CALL_STACK_TRACE_DISABLE macro	270	MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES	277
MXD_ASSERT_CALL_STACK_TRACE_OVERRIDE	270	MXD_CAP_SUBALLOCATOR_DEFAULT_MEMORY_BLOCK_SIZE_IN_BYTES	macro 277
MXD_ASSERT_CALL_STACK_TRACE_OVERRIDE macro	270	MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT	277
MXD_ASSERT_DISABLE_DEBUG_BREAK	271	MXD_CAP_SUBALLOCATOR_ENABLE_SUPPORT macro	277
MXD_ASSERT_DISABLE_DEBUG_BREAK macro	271	MXD_CAP_SUBALLOCATOR_STATISTICS_ENABLE_SUPPORT	278
MXD_ASSERT_DISABLE_OUTPUT_FILENAME	271	MXD_CAP_SUBALLOCATOR_STATISTICS_ENABLE_SUPPORT macro	278
MXD_ASSERT_DISABLE_OUTPUT_FILENAME macro	271	MXD_CFILE_TRACES_ENABLE_SUPPORT	278
MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER	271	MXD_CFILE_TRACES_ENABLE_SUPPORT macro	278
MXD_ASSERT_DISABLE_OUTPUT_LINENUMBER macro	271	MXD_CMARSHALER_ENABLE_DEBUG	278
MXD_ASSERT_ENABLE_ALL	272	MXD_CMARSHALER_ENABLE_DEBUG macro	278
MXD_ASSERT_ENABLE_ALL macro	272	MXD_COMPILER_DIAB	278
MXD_ASSERT_ENABLE_RT	272	MXD_COMPILER_DIAB macro	278
MXD_ASSERT_ENABLE_RT macro	272	MXD_COMPILER_GNU_GCC	279
MXD_ASSERT_ENABLE_STD	272	MXD_COMPILER_GNU_GCC macro	279
MXD_ASSERT_ENABLE_SUPPORT	272	MXD_COMPILER_MS_VC	279
MXD_ASSERT_ENABLE_SUPPORT macro	272	MXD_COMPILER_MS_VC macro	279
MXD_ASSERT_FAIL_OVERRIDE	272	MXD_COMPILER_TI_CL6X	279
MXD_ASSERT_FAIL_OVERRIDE macro	272	MXD_COMPILER_TI_CL6X macro	279
MXD_ASSERT_FINAL_BEHAVIOR_DISABLE	273	MXD_CRYPTO_AES_CLASSNAME	280
MXD_ASSERT_FINAL_BEHAVIOR_DISABLE macro	273	MXD_CRYPTO_AES_CLASSNAME macro	280
MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL	273	MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELATION	279
MXD_ASSERT_FINAL_BEHAVIOR_IS_FATAL macro	273	MXD_CRYPTO_AES_CORE_ALLOW_CONSTANTS_TABLE_RELATION macro	279
MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE	273	MXD_CRYPTO_AES_CORE_UNROLL	280
MXD_ASSERT_FINAL_BEHAVIOR_OVERRIDE macro	273	MXD_CRYPTO_AES_CORE_UNROLL macro	280
MXD_ASSERT_TRACE_DISABLE	273	MXD_CRYPTO_AES_CTR_MODE_ONLY	280
MXD_ASSERT_TRACE_DISABLE macro	273	MXD_CRYPTO_AES_CTR_MODE_ONLY macro	280
MXD_ASSERT_TRACE_OVERRIDE	274	MXD_CRYPTO_AES_INCLUDE	280
MXD_ASSERT_TRACE_OVERRIDE macro	274	MXD_CRYPTO_AES_INCLUDE macro	280
MXD_astMEMORY_ALLOCATOR_POOL_INFO	274	MXD_CRYPTO_AES_MITOSFW	280
MXD_astMEMORY_ALLOCATOR_POOL_INFO macro	274	MXD_CRYPTO_AES_MITOSFW macro	280
MXD_astPOSIX_THREAD_SCHEDULING_INFO	275	MXD_CRYPTO_AES_MOCANA_SS	280
MXD_astPOSIX_THREAD_SCHEDULING_INFO macro	275	MXD_CRYPTO_AES_MOCANA_SS macro	280
MXD_astVXWORKS_THREAD_SCHEDULING_INFO	275	MXD_CRYPTO_AES_NONE	280
MXD_astWINDOWS_THREAD_SCHEDULING_INFO	275	MXD_CRYPTO_AES_OPENSSL	280
MXD_astWINDOWS_THREAD_SCHEDULING_INFO macro	275		
MXD_ATOMIC_NATIVE_ENABLE_SUPPORT	276		
MXD_ATOMIC_NATIVE_ENABLE_SUPPORT macro	276		
MXD_BIG_ENDIAN	276		
MXD_BIG_ENDIAN macro	276		
MXD_CAA TREE_ENABLE_DEBUG	277		
MXD_CAA TREE_ENABLE_DEBUG macro	277		

MXD_CRYPTO_AES_OPENSSL macro 280
MXD_CRYPTO_AES_OVERRIDE 280
MXD_CRYPTO_AES_OVERRIDE macro 280
MXD_CRYPTO_ALL_MITOSFW 281
MXD_CRYPTO_ALL_MITOSFW macro 281
MXD_CRYPTO_ALL_MOCANA_SS 281
MXD_CRYPTO_ALL_MOCANA_SS macro 281
MXD_CRYPTO_ALL_NONE 281
MXD_CRYPTO_ALL_OPENSSL 281
MXD_CRYPTO_ALL_OPENSSL macro 281
MXD_CRYPTO_ALL_OVERRIDE 281
MXD_CRYPTO_ALL_OVERRIDE macro 281
MXD_CRYPTO_BASE64_MITOSFW 282
MXD_CRYPTO_BASE64_MITOSFW macro 282
MXD_CRYPTO_BASE64_NONE 282
MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME 282
MXD_CRYPTO_DIFFIEHELLMAN_CLASSNAME macro 282
MXD_CRYPTO_DIFFIEHELLMAN_INCLUDE 282
MXD_CRYPTO_DIFFIEHELLMAN_INCLUDE macro 282
MXD_CRYPTO_DIFFIEHELLMAN_MOCANA_SS 282
MXD_CRYPTO_DIFFIEHELLMAN_MOCANA_SS macro 282
MXD_CRYPTO_DIFFIEHELLMAN_NONE 282
MXD_CRYPTO_DIFFIEHELLMAN_OPENSSL 282
MXD_CRYPTO_DIFFIEHELLMAN_OPENSSL macro 282
MXD_CRYPTO_DIFFIEHELLMAN_OVERRIDE 282
MXD_CRYPTO_DIFFIEHELLMAN_OVERRIDE macro 282
MXD_CRYPTO_MD5_CLASSNAME 282
MXD_CRYPTO_MD5_CLASSNAME macro 282
MXD_CRYPTO_MD5_INCLUDE 282
MXD_CRYPTO_MD5_INCLUDE macro 282
MXD_CRYPTO_MD5_MITOSFW 282
MXD_CRYPTO_MD5_MITOSFW macro 282
MXD_CRYPTO_MD5_MOCANA_SS 282
MXD_CRYPTO_MD5_MOCANA_SS macro 282
MXD_CRYPTO_MD5_NONE 282
MXD_CRYPTO_MD5_OPENSSL 282
MXD_CRYPTO_MD5_OPENSSL macro 282
MXD_CRYPTO_MD5_OVERRIDE 282
MXD_CRYPTO_MD5_OVERRIDE macro 282
MXD_CRYPTO_MD5MAC_CLASSNAME 283
MXD_CRYPTO_MD5MAC_CLASSNAME macro 283
MXD_CRYPTO_MD5MAC_INCLUDE 283
MXD_CRYPTO_MD5MAC_INCLUDE macro 283
MXD_CRYPTO_MD5MAC_MITOSFW 283
MXD_CRYPTO_MD5MAC_MITOSFW macro 283
MXD_CRYPTO_MD5MAC_MOCANA_SS 283
MXD_CRYPTO_MD5MAC_MOCANA_SS macro 283
MXD_CRYPTO_MD5MAC_NONE 283
MXD_CRYPTO_MD5MAC_OPENSSL 283
MXD_CRYPTO_MD5MAC_OPENSSL macro 283
MXD_CRYPTO_MD5MAC_OVERRIDE 283
MXD_CRYPTO_MD5MAC_OVERRIDE macro 283
MXD_CRYPTO_PRIVATEKEY_CLASSNAME 284
MXD_CRYPTO_PRIVATEKEY_CLASSNAME macro 284
MXD_CRYPTO_PRIVATEKEY_INCLUDE 284
MXD_CRYPTO_PRIVATEKEY_INCLUDE macro 284
MXD_CRYPTO_PRIVATEKEY_MITOSFW 284
MXD_CRYPTO_PRIVATEKEY_MITOSFW macro 284
MXD_CRYPTO_PRIVATEKEY_MOCANA_SS 284
MXD_CRYPTO_PRIVATEKEY_MOCANA_SS macro 284
MXD_CRYPTO_PRIVATEKEY_NONE 284
MXD_CRYPTO_PRIVATEKEY_OPENSSL 284
MXD_CRYPTO_PRIVATEKEY_OPENSSL macro 284
MXD_CRYPTO_PRIVATEKEY_OVERRIDE 284
MXD_CRYPTO_PRIVATEKEY_OVERRIDE macro 284
MXD_CRYPTO_PUBLICKEY_CLASSNAME 284
MXD_CRYPTO_PUBLICKEY_CLASSNAME macro 284
MXD_CRYPTO_PUBLICKEY_INCLUDE 284
MXD_CRYPTO_PUBLICKEY_INCLUDE macro 284
MXD_CRYPTO_PUBLICKEY_MITOSFW 284
MXD_CRYPTO_PUBLICKEY_MITOSFW macro 284
MXD_CRYPTO_PUBLICKEY_MOCANA_SS 284
MXD_CRYPTO_PUBLICKEY_MOCANA_SS macro 284
MXD_CRYPTO_PUBLICKEY_NONE 284
MXD_CRYPTO_PUBLICKEY_OPENSSL 284
MXD_CRYPTO_PUBLICKEY_OPENSSL macro 284
MXD_CRYPTO_PUBLICKEY_OVERRIDE 284
MXD_CRYPTO_PUBLICKEY_OVERRIDE macro 284
MXD_CRYPTO_RSA_CLASSNAME 285
MXD_CRYPTO_RSA_CLASSNAME macro 285
MXD_CRYPTO_RSA_INCLUDE 285
MXD_CRYPTO_RSA_INCLUDE macro 285
MXD_CRYPTO_RSA_MITOSFW 285
MXD_CRYPTO_RSA_MITOSFW macro 285
MXD_CRYPTO_RSA_MOCANA_SS 285
MXD_CRYPTO_RSA_MOCANA_SS macro 285

MXD_CRYPTO_RSA_NONE 285	MXD_CRYPTO_SHA1MAC_CLASSNAME macro 287
MXD_CRYPTO_RSA_OPENSSL 285	MXD_CRYPTO_SHA1MAC_INCLUDE 287
MXD_CRYPTO_RSA_OPENSSL macro 285	MXD_CRYPTO_SHA1MAC_INCLUDE macro 287
MXD_CRYPTO_RSA_OVERRIDE 285	MXD_CRYPTO_SHA1MAC_MITOSFW 287
MXD_CRYPTO_RSA_OVERRIDE macro 285	MXD_CRYPTO_SHA1MAC_MITOSFW macro 287
MXD_CRYPTO_SECUREPRNG_CLASSNAME 285	MXD_CRYPTO_SHA1MAC_NONE 287
MXD_CRYPTO_SECUREPRNG_CLASSNAME macro 285	MXD_CRYPTO_SHA1MAC_OPENSSL 287
MXD_CRYPTO_SECUREPRNG_INCLUDE 285	MXD_CRYPTO_SHA1MAC_OPENSSL macro 287
MXD_CRYPTO_SECUREPRNG_INCLUDE macro 285	MXD_CRYPTO_SHA1MAC_OVERRIDE 287
MXD_CRYPTO_SECUREPRNG_MITOSFW 285	MXD_CRYPTO_SHA1MAC_OVERRIDE macro 287
MXD_CRYPTO_SECUREPRNG_MITOSFW macro 285	MXD_CRYPTO_SHA2_CLASSNAME 288
MXD_CRYPTO_SECUREPRNG_MOCANA_SS 285	MXD_CRYPTO_SHA2_CLASSNAME macro 288
MXD_CRYPTO_SECUREPRNG_MOCANA_SS macro 285	MXD_CRYPTO_SHA2_INCLUDE 288
MXD_CRYPTO_SECUREPRNG_NONE 285	MXD_CRYPTO_SHA2_INCLUDE macro 288
MXD_CRYPTO_SECUREPRNG_OVERRIDE 285	MXD_CRYPTO_SHA2_MITOSFW 288
MXD_CRYPTO_SECUREPRNG_OVERRIDE macro 285	MXD_CRYPTO_SHA2_MITOSFW macro 288
MXD_CRYPTO_SECURESEED_CLASSNAME 286	MXD_CRYPTO_SHA2_NONE 288
MXD_CRYPTO_SECURESEED_CLASSNAME macro 286	MXD_CRYPTO_SHA2_OPENSSL 288
MXD_CRYPTO_SECURESEED_INCLUDE 286	MXD_CRYPTO_SHA2_OPENSSL macro 288
MXD_CRYPTO_SECURESEED_INCLUDE macro 286	MXD_CRYPTO_SHA2_OVERRIDE 288
MXD_CRYPTO_SECURESEED_MITOSFW 286	MXD_CRYPTO_SHA2_OVERRIDE macro 288
MXD_CRYPTO_SECURESEED_MITOSFW macro 286	MXD_CRYPTO_SHA2MAC_CLASSNAME 288
MXD_CRYPTO_SECURESEED_MOCANA_SS 286	MXD_CRYPTO_SHA2MAC_CLASSNAME macro 288
MXD_CRYPTO_SECURESEED_MOCANA_SS macro 286	MXD_CRYPTO_SHA2MAC_INCLUDE 288
MXD_CRYPTO_SECURESEED_NONE 286	MXD_CRYPTO_SHA2MAC_INCLUDE macro 288
MXD_CRYPTO_SECURESEED_OVERRIDE 286	MXD_CRYPTO_SHA2MAC_MITOSFW 288
MXD_CRYPTO_SECURESEED_OVERRIDE macro 286	MXD_CRYPTO_SHA2MAC_MITOSFW macro 288
MXD_CRYPTO_SHA1_CLASSNAME 287	MXD_CRYPTO_SHA2MAC_NONE 288
MXD_CRYPTO_SHA1_CLASSNAME macro 287	MXD_CRYPTO_SHA2MAC_OPENSSL 288
MXD_CRYPTO_SHA1_DISABLE_LONG_MESSAGES 286	MXD_CRYPTO_SHA2MAC_OPENSSL macro 288
MXD_CRYPTO_SHA1_DISABLE_LONG_MESSAGES macro 286	MXD_CRYPTO_SHA2MAC_OVERRIDE 288
MXD_CRYPTO_SHA1_INCLUDE 287	MXD_CRYPTO_SHA2MAC_OVERRIDE macro 288
MXD_CRYPTO_SHA1_INCLUDE macro 287	MXD_CRYPTO_UUIDGENERATOR_CLASSNAME 286
MXD_CRYPTO_SHA1_MITOSFW 287	MXD_CRYPTO_UUIDGENERATOR_CLASSNAME macro 286
MXD_CRYPTO_SHA1_MITOSFW macro 287	MXD_CRYPTO_UUIDGENERATOR_INCLUDE 286
MXD_CRYPTO_SHA1_MOCANA_SS 287	MXD_CRYPTO_UUIDGENERATOR_INCLUDE macro 286
MXD_CRYPTO_SHA1_MOCANA_SS macro 287	MXD_CRYPTO_UUIDGENERATOR_MITOSFW 289
MXD_CRYPTO_SHA1_NONE 287	MXD_CRYPTO_UUIDGENERATOR_MITOSFW macro 289
MXD_CRYPTO_SHA1_OPENSSL 287	MXD_CRYPTO_UUIDGENERATOR_NONE 289
MXD_CRYPTO_SHA1_OPENSSL macro 287	MXD_CRYPTO_UUIDGENERATOR_OVERRIDE 286
MXD_CRYPTO_SHA1_OVERRIDE 287	MXD_CRYPTO_UUIDGENERATOR_OVERRIDE macro 286
MXD_CRYPTO_SHA1_OVERRIDE macro 287	MXD_CSTRING_MINIMUM_BUFFER 290
MXD_CRYPTO_SHA1MAC_CLASSNAME 287	MXD_CSTRING_MINIMUM_BUFFER macro 290

MXD_DATA_MODEL_ILP32 290	MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT 295
MXD_DATA_MODEL_ILP32 macro 290	MXD_MEMORY_ALLOCATOR_BOUND_CHECK_ENABLE_SUPPORT macro 295
MXD_DATA_MODEL_LP64 290	MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT 296
MXD_DATA_MODEL_LP64 macro 290	MXD_MEMORY_ALLOCATOR_ENABLE_SUPPORT macro 296
MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET 291	MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT 296
MXD_DEFAULT_THREAD_STACK_INFO_BUFFER_OFFSET macro 291	MXD_MEMORY_ALLOCATOR_EXTRA_INFORMATION_ENABLE_SUPPORT macro 296
MXD_DEFAULT_THREAD_STACK_SIZE 291	MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT 296
MXD_DEFAULT_THREAD_STACK_SIZE macro 291	MXD_MEMORY_ALLOCATOR_MEMORY_POOL_ENABLE_SUPPORT macro 296
MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE 291	MXD_MEMORY_ALLOCATOR_MEMORY_TRACKING_ENABLE_SUPPORT 297
MXD_DISABLE_EXTERNAL_ASSERT_OVERRIDE macro 291	MXD_MEMORY_ALLOCATOR_PROTECTION_ENABLE_SUPPORT 297
MXD_ECOM_ENABLE_SUPPORT 292	MXD_MEMORY_ALLOCATOR_STATISTICS_ENABLE_SUPPORT 297
MXD_ECOM_ENABLE_SUPPORT macro 292	MXD_MINIMAL_ALIGNMENT_IN_BYTES 298
MXD_ENABLE_NAMESPACE 317	MXD_MINIMAL_ALIGNMENT_IN_BYTES macro 298
MXD_ENABLE_NAMESPACE macro 317	MXD_NETWORK_ENABLE_SUPPORT 298
MXD_ENUM_ENABLE_SUPPORT 292	MXD_OPENSSL_FIPS_140_2_ENABLE_SUPPORT 281
MXD_ENUM_ENABLE_SUPPORT macro 292	MXD_OPENSSL_FIPS_140_2_ENABLE_SUPPORT macro 281
MXD_ENUM_NAPTR_MAX_NON_TERMINAL 292	MXD_OS_LINUX 298
MXD_ENUM_NAPTR_MAX_NON_TERMINAL macro 292	MXD_OS_LINUX macro 298
MXD_EVENT_NOTIFIER_ENABLE_SUPPORT 292	MXD_OS_NONE 318
MXD_EVENT_NOTIFIER_ENABLE_SUPPORT macro 292	MXD_OS_NUCLEUS 298
MXD_FAULT_HANDLER_ENABLE_SUPPORT 292	MXD_OS_NUCLEUS macro 298
MXD_FAULT_HANDLER_ENABLE_SUPPORT macro 292	MXD_OS_VXWORKS 298
MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_EAKED_MEMORY_BLOCKS 293	
MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_EAKED_MEMORY_BLOCKS macro 293	
MXD_GNS 56	
MXD_INCLUDE_NEW 293	
MXD_INCLUDE_NEW macro 293	
MXD_IPV6_ENABLE_SUPPORT 294	
MXD_IPV6_ENABLE_SUPPORT macro 294	
MXD_IPV6_SCOPE_ID_MANDATORY_IN_ALL_ADDRESSES 294	
MXD_IPV6_SCOPE_ID_MANDATORY_IN_ALL_ADDRESSES macro 294	
MXD_KERBEROS_ENABLE_SUPPORT 294	
MXD_KERBEROS_ENABLE_SUPPORT macro 294	
MXD_LIB_GNU_LIBC 295	
MXD_LIB_GNU_LIBC macro 295	
MXD_LIB_GNU_UCLIBC 295	
MXD_LIB_GNU_UCLIBC macro 295	
MXD_LITTLE_ENDIAN 295	
MXD_LITTLE_ENDIAN macro 295	

MXD_OS_VXWORKS macro 298	MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME 301
MXD_OS_WINDOWS 299	MXD_PKI_CCERTIFICATESUBJECT_CLASSNAME macro 301
MXD_OS_WINDOWS macro 299	MXD_PKI_CCERTIFICATESUBJECT_INCLUDE 301
MXD_OS_WINDOWS_CE 299	MXD_PKI_CCERTIFICATESUBJECT_INCLUDE macro 301
MXD_OS_WINDOWS_CE macro 299	MXD_PKI_CERTIFICATECHAIN_INCLUDE 301
MXD_OS_WINDOWS_ENABLE_GQOS_QOS 300	MXD_PKI_CERTIFICATECHAIN_INCLUDE macro 301
MXD_OS_WINDOWS_ENABLE_GQOS_QOS macro 300	MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME 301
MXD_OS_WINDOWS_ENABLE_TC_QOS 300	MXD_PKI_CEXTENDEDKEYUSAGE_CLASSNAME macro 301
MXD_OS_WINDOWS_ENABLE_TC_QOS macro 300	MXD_PKI_CEXTENDEDKEYUSAGE_INCLUDE 301
MXD_PKG_ID_OVERRIDE 300	MXD_PKI_CEXTENDEDKEYUSAGE_INCLUDE macro 301
MXD_PKG_ID_OVERRIDE macro 300	MXD_PKI_CKEYUSAGE_CLASSNAME 301
MXD_PKI_CALTERNATENAME_CLASSNAME 301	MXD_PKI_CKEYUSAGE_CLASSNAME macro 301
MXD_PKI_CALTERNATENAME_CLASSNAME macro 301	MXD_PKI_CKEYUSAGE_INCLUDE 301
MXD_PKI_CALTERNATENAME_INCLUDE 301	MXD_PKI_CKEYUSAGE_INCLUDE macro 301
MXD_PKI_CALTERNATENAME_INCLUDE macro 301	MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME 301
MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME 301	MXD_PKI_CSUBJECTKEYIDENTIFIER_CLASSNAME macro 301
MXD_PKI_CAUTHORITYKEYIDENTIFIER_CLASSNAME macro 301	MXD_PKI_CSUBJECTKEYIDENTIFIER_INCLUDE 301
MXD_PKI_CAUTHORITYKEYIDENTIFIER_INCLUDE 301	MXD_PKI_CSUBJECTKEYIDENTIFIER_INCLUDE macro 301
MXD_PKI_CAUTHORITYKEYIDENTIFIER_INCLUDE macro 301	MXD_PKI_MOCANA_SS 301
MXD_PKI_CBASICCONSTRAINTS_CLASSNAME 301	MXD_PKI_MOCANA_SS macro 301
MXD_PKI_CBASICCONSTRAINTS_CLASSNAME macro 301	MXD_PKI_NONE 301
MXD_PKI_CBASICCONSTRAINTS_INCLUDE 301	MXD_PKI_OPENSSL 301
MXD_PKI_CBASICCONSTRAINTS_INCLUDE macro 301	MXD_PKI_OPENSSL macro 301
MXD_PKI_CCERTIFICATE_CLASSNAME 301	MXD_PKI_OVERRIDE 301
MXD_PKI_CCERTIFICATE_CLASSNAME macro 301	MXD_PKI_OVERRIDE macro 301
MXD_PKI_CCERTIFICATE_INCLUDE 301	MXD_PORTABLE_RESOLVER_ENABLE_SUPPORT 302
MXD_PKI_CCERTIFICATE_INCLUDE macro 301	MXD_PORTABLE_RESOLVER_ENABLE_SUPPORT macro 302
MXD_PKI_CCERTIFICATECHAIN_CLASSNAME 301	MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS 303
MXD_PKI_CCERTIFICATECHAIN_CLASSNAME macro 301	MXD_PORTABLE_RESOLVER_MAX_RETRANSMISSIONS macro 303
MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME 301	MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS 303
MXD_PKI_CCERTIFICATECHAINVALIDATION_CLASSNAME macro 301	MXD_PORTABLE_RESOLVER_RETRANSMISSION_TIMEOUT_MS macro 303
MXD_PKI_CCERTIFICATECHAINVALIDATION_INCLUDE 301	MXD_POST_CONFIG 303
MXD_PKI_CCERTIFICATECHAINVALIDATION_INCLUDE macro 301	MXD_POST_CONFIG macro 303
MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME 301	MXD_POST_FRAMEWORKCFG 303
MXD_PKI_CCERTIFICATEEXTENSION_CLASSNAME macro 301	MXD_POST_FRAMEWORKCFG macro 303
MXD_PKI_CCERTIFICATEEXTENSION_INCLUDE 301	MXD_REGEXP_ENABLE_SUPPORT 304
MXD_PKI_CCERTIFICATEEXTENSION_INCLUDE macro 301	MXD_REGEXP_ENABLE_SUPPORT macro 304
MXD_PKI_CCERTIFICATEISSUER_CLASSNAME 301	MXD_RESOLVER_CACHE_CAPACITY 304
MXD_PKI_CCERTIFICATEISSUER_CLASSNAME macro 301	MXD_RESOLVER_CACHE_CAPACITY macro 304
MXD_PKI_CCERTIFICATEISSUER_INCLUDE 301	MXD_RESOLVER_CACHE_ENABLE_SUPPORT 304
MXD_PKI_CCERTIFICATEISSUER_INCLUDE macro 301	

MXD_RESOLVER_CACHE_ENABLE_SUPPORT macro 304	MXD_TLS_CTLS_INCLUDE macro 308
MXD_RESOLVER_CACHE_NEGATIVE_MAX_TTL_S 304	MXD_TLS_CTLSESSION_CLASSNAME 308
MXD_RESOLVER_CACHE_NEGATIVE_MAX_TTL_S macro 304	MXD_TLS_CTLSESSION_CLASSNAME macro 308
MXD_RESOLVER_CACHE_POSITIVE_MAX_TTL_S 305	MXD_TLS_CTLSESSION_INCLUDE 308
MXD_RESOLVER_CACHE_POSITIVE_MAX_TTL_S macro 305	MXD_TLS_CTLSESSION_INCLUDE macro 308
MXD_RESOLVER_HOST_FILE_ENABLE_SUPPORT 305	MXD_TLS_MOCANA_SS 308
MXD_RESOLVER_HOST_FILE_ENABLE_SUPPORT macro 305	MXD_TLS_MOCANA_SS macro 308
MXD_RESULT_ENABLE_ALL_ERROR_MESSAGES 305	MXD_TLS_NONE 308
MXD_RESULT_ENABLE_ALL_ERROR_MESSAGES macro 305	MXD_TLS_OPENSSL 308
MXD_RESULT_ENABLE_ERROR_MESSAGES 305	MXD_TLS_OPENSSL macro 308
MXD_RESULT_ENABLE_ERROR_MESSAGES macro 305	MXD_TRACE_BACKTRACE_CAPACITY 309
MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES 306	MXD_TRACE_BACKTRACE_CAPACITY macro 309
MXD_RESULT_ENABLE_MITOSFW_ERROR_MESSAGES macro 306	MXD_TRACE_BUFFER_CAPACITY 309
MXD_RESULT_ENABLE_SHARED_ERROR_MESSAGES 306	MXD_TRACE_BUFFER_CAPACITY macro 309
MXD_RESULT_ENABLE_SHARED_ERROR_MESSAGES macro 306	MXD_TRACE_BUFFER_OVERRIDE 313
MXD_SERVICING_THREAD_ENABLE_SUPPORT 306	MXD_TRACE_BUFFER_OVERRIDE macro 313
MXD_SERVICING_THREAD_ENABLE_SUPPORT macro 306	MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE 309
MXD_SERVICING_THREAD_MAX_CONSECUTIVE_ITERATIONS 306	MXD_TRACE_CALLSTACK_HANDLER_OVERRIDE macro 309
MXD_SERVICING_THREAD_MAX_CONSECUTIVE_ITERATIONS macro 306	MXD_TRACE_EMPTY_MEMORY_QUEUE_ON_FINALIZE 310
MXD_SNTP_CLIENT_ENABLE_SUPPORT 307	MXD_TRACE_EMPTY_MEMORY_QUEUE_ON_FINALIZE macro 310
MXD_SNTP_CLIENT_ENABLE_SUPPORT macro 307	MXD_TRACE_ENABLE_ALL 314
MXD_STRING_DISABLE_REFCOUNT 307	MXD_TRACE_ENABLE_ALL macro 314
MXD_STRING_DISABLE_REFCOUNT macro 307	MXD_TRACE_ENABLE_MACRO_CALLSTACK 314
MXD_SYSTEM_MEMORY_SIZE 307	MXD_TRACE_ENABLE_MACRO_CALLSTACK macro 314
MXD_SYSTEM_MEMORY_SIZE macro 307	MXD_TRACE_FORMAT_HANDLER_OVERRIDE 310
MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT 307	MXD_TRACE_FORMAT_HANDLER_OVERRIDE macro 310
MXD_THREAD_FIXED_STACK_SIZE_ENABLE_SUPPORT macro 307	MXD_TRACE_MAX_LEVEL 314
MXD_THREAD_STACK_INFO_ENABLE_SUPPORT 308	MXD_TRACE_MAX_LEVEL macro 314
MXD_THREAD_STACK_INFO_ENABLE_SUPPORT macro 308	MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS 310
MXD_TIME_ENABLE_SUPPORT 308	MXD_TRACE_MAX_NB_OF_OUTPUT_HANDLERS macro 310
MXD_TIME_ENABLE_SUPPORT macro 308	MXD_TRACE_MESSAGES_PER_PERIOD 310
MXD_TLS_CASYNCTLSSERVERSOCKET_INCLUDE 308	MXD_TRACE_MESSAGES_PER_PERIOD macro 310
MXD_TLS_CASYNCTLSSERVERSOCKET_INCLUDE macro 308	MXD_TRACE_NODES_ENABLED_AT_REGISTRATION 311
MXD_TLS_CASYNCTLSSOCKET_INCLUDE 308	MXD_TRACE_NODES_ENABLED_AT_REGISTRATION macro 311
MXD_TLS_CASYNCTLSSOCKET_INCLUDE macro 308	MXD_TRACE_OUTPUT_HANDLER_OVERRIDE 311
MXD_TLS_CTLS_CLASSNAME 308	MXD_TRACE_OUTPUT_HANDLER_OVERRIDE macro 311
MXD_TLS_CTLS_CLASSNAME macro 308	MXD_TRACE_PERIOD_MS 312
MXD_TLS_CTLS_INCLUDE 308	MXD_TRACE_PERIOD_MS macro 312
	MXD_TRACE_PROGNAME 312
	MXD_TRACE_PROGNAME macro 312
	MXD_TRACE_QUEUE_SIZE 312
	MXD_TRACE_QUEUE_SIZE macro 312

MXD_TRACE_SEPARATOR 312
MXD_TRACE_SEPARATOR macro 312
MXD_TRACE_THREAD_NAME 313
MXD_TRACE_THREAD_NAME macro 313
MXD_TRACE_THREAD_PRIORITY 313
MXD_TRACE_THREAD_PRIORITY macro 313
MXD_TRACE_THREAD_STACK_SIZE 313
MXD_TRACE_THREAD_STACK_SIZE macro 313
MXD_TRACE_USE_DYNAMIC_ALLOC_BUFFER 313
MXD_TRACE_USE_EXTERNAL_THREAD 314
MXD_TRACE_USE_EXTERNAL_THREAD macro 314
MXD_TRACE_USE_MEMORY_QUEUE_BUFFER 313
MXD_TRACE_USE_MEMORY_QUEUE_BUFFER macro 313
MXD_TRACE_USE_STACK_BUFFER 313
MXD_TRACE_USE_STACK_BUFFER macro 313
MXD_TRACE0_ENABLE_SUPPORT 314
MXD_TRACE1_ENABLE_SUPPORT 314
MXD_TRACE1_ENABLE_SUPPORT macro 314
MXD_TRACE2_ENABLE_SUPPORT 314
MXD_TRACE2_ENABLE_SUPPORT macro 314
MXD_TRACE3_ENABLE_SUPPORT 314
MXD_TRACE3_ENABLE_SUPPORT macro 314
MXD_TRACE4_ENABLE_SUPPORT 314
MXD_TRACE4_ENABLE_SUPPORT macro 314
MXD_TRACE5_ENABLE_SUPPORT 314
MXD_TRACE5_ENABLE_SUPPORT macro 314
MXD_TRACE6_ENABLE_SUPPORT 314
MXD_TRACE6_ENABLE_SUPPORT macro 314
MXD_TRACE7_ENABLE_SUPPORT 314
MXD_TRACE7_ENABLE_SUPPORT macro 314
MXD_TRACE8_ENABLE_SUPPORT 314
MXD_TRACE9_ENABLE_SUPPORT 314
MXD_TRACE9_ENABLE_SUPPORT macro 314
MXD_TRACEX_ENABLE_SUPPORT 314
MXD_TRACEX_ENABLE_SUPPORT macro 314
MXD_uCSTRING_BLOCK_LENGTH 315
MXD_uCSTRING_BLOCK_LENGTH macro 315
MXD_UINT_MAX 315
MXD_UINT_MAX macro 315
MXD_VXWORKS_VERSION 316
MXD_VXWORKS_VERSION macro 316
MXD_XML_DEPRECATED_ENABLE_SUPPORT 316
MXD_XML_ENABLE_SUPPORT 316
MXD_XML_ENABLE_SUPPORT macro 316
MXD_XML_PARSER_EXPAT_ENABLE_SUPPORT 316
MXD_XML_PARSER_EXPAT_ENABLE_SUPPORT macro 316
MxIntToString 20
MxIntToString function 20
MxRemove 507
MxRemove function 507
MxRename 507
MxRename function 507
MxSnprintf 21
MxSnprintf function 21
MxStrErrorReentrant 21
MxStrErrorReentrant function 21
MxStringCaseCompare 21
MxStringCaseCompare function 21
MxStringToInt 22
MxStringToInt function 22
MxStringToUint 22, 23
MxStringToUint function 22, 23
mxt_clsid 426
mxt_clsid type 426
mxt_fd 521
mxt_fd type 521
mxt_iid 426
mxt_iid type 426
mxt_opaque 87
mxt_opaque type 87
mxt_PFNAssertCallStackTraceHandler 87
mxt_PFNAssertCallStackTraceHandler type 87
mxt_PFNAssertFailHandler 88
mxt_PFNAssertFailHandler type 88
mxt_PFNAssertFinalBehaviorHandler 88
mxt_PFNAssertFinalBehaviorHandler type 88
mxt_PFNAssertTraceHandler 88
mxt_PFNAssertTraceHandler type 88
mxt_pfnCreateEComInstance 426
mxt_pfnCreateEComInstance type 426
mxt_pfnEventObserver 89
mxt_pfnEventObserver type 89
mxt_PFNTraceCallStackHandler 89
mxt_PFNTraceCallStackHandler type 89

mxt_PFNTraceFormatHandler	89	MxTraceEnableNode	27	
mxt_PFNTraceFormatHandler	type	89	MxTraceEnableNode function	27
mxt_PFNTraceGenericFormatHandler	91	MxTraceEnableNodeLevel	27	
mxt_PFNTraceGenericFormatHandler	type	91	MxTraceEnableNodeLevel function	27
mxt_PFNTraceOutputHandler	92	MxTraceEnableNodeMaxLevels	28	
mxt_PFNTraceOutputHandler	type	92	MxTraceEnableNodeMaxLevels function	28
mxt_result	92	MxTraceEnableSyslogFields	28	
mxt_result	type	92	MxTraceEnableSyslogFields function	28
mxt_UNALIGNED_INT16	93	MxTraceFieldIsEnabled	29	
mxt_UNALIGNED_INT32	93	MxTraceFieldIsEnabled function	29	
mxt_UNALIGNED_INT32	type	93	MxTraceGetLevelsEnabled	29
mxt_UNALIGNED_INT64	93	MxTraceGetLevelsEnabled function	29	
mxt_UNALIGNED_INT64	type	93	MxTraceGetNodeLevelsEnabled	29
mxt_UNALIGNED_UINT16	93	MxTraceGetNodeLevelsEnabled function	29	
mxt_UNALIGNED_UINT16	type	93	MxTracePrintTree	30
mxt_UNALIGNED_UINT32	93	MxTracePrintTree function	30	
mxt_UNALIGNED_UINT32	type	93	MxTraceRegisterNode	30
mxt_UNALIGNED_UINT64	93	MxTraceRegisterNode function	30	
mxt_UNALIGNED_UINT64	type	93	MxTraceRemoveOutputHandler	31
MxTraceAddOutputHandler	23	MxTraceRemoveOutputHandler function	31	
MxTraceAddOutputHandler	function	23	MxTraceSetHostNameField	31
MxTraceCallOutputHandler	508	MxTraceSetHostNameField function	31	
MxTraceCallOutputHandler	function	508	MxTraceSetNewCallStackHandler	31
MxTraceDisableAllFields	24	MxTraceSetNewCallStackHandler function	31	
MxTraceDisableAllFields	function	24	MxTraceSetNewFormatHandler	32
MxTraceDisableField	24	MxTraceSetNewFormatHandler function	32	
MxTraceDisableField	function	24	MxTraceSetNewOutputHandler	32
MxTraceDisableLevel	24	MxTraceSetNewOutputHandler function	32	
MxTraceDisableLevel	function	24	MxTraceSetSysTimeFormatHandler	32
MxTraceDisableNode	24	MxTraceSetSysTimeFormatHandler function	32	
MxTraceDisableNode	function	24	MxTraceSetThreadMessagesPerPeriod	508
MxTraceDisableNodeLevel	25	MxTraceSetThreadMessagesPerPeriod function	508	
MxTraceDisableNodeLevel	function	25	MxTraceSetThreadPeriodMs	509
MxTraceEnableAllFields	25	MxTraceSetThreadPeriodMs function	509	
MxTraceEnableAllFields	function	25	MxTraceSetTimeFormatHandler	33
MxTraceEnableDefaultFields	26	MxTraceSetTimeFormatHandler function	33	
MxTraceEnableDefaultFields	function	26	MxTraceToDebugger	33
MxTraceEnableField	26	MxTraceToDebugger function	33	
MxTraceEnableField	function	26	MxTraceToStderr	33
MxTraceEnableLevel	26	MxTraceToStderr function	33	
MxTraceEnableLevel	function	26	MxTraceToStdout	34
MxTraceEnableMaxLevels	26	MxTraceToStdout function	34	
MxTraceEnableMaxLevels	function	26	MxTraceUnregisterNode	34

MxTraceUnregisterNode function 34

MxUintToString 35

MxUintToString function 35

MxVsnprintf 35

MxVsnprintf function 35

N

Network 522

NotifyEventObservers 36

NotifyEventObservers function 36

O

OUT 39

OUT macro 39

P

Parameters qualifier 39

Pki 663

Pki configuration macros 301

PostMxConfig.h 317

PreMxConfig.h 317

R

RegExp (Regular Expressions) 738

RegisterECom 422

RegisterECom function 422

RegisterEventObserver 36

RegisterEventObserver function 36

resFC_CRITICAL enumeration member 13

resFC_FIRST enumeration member 13

resFC_LAST enumeration member 13

resFE_ABORT enumeration member 13

resFE_ACCESS_DENIED enumeration member 13

resFE_DUPLICATE enumeration member 13

resFE_FAIL enumeration member 13

resFE_FIRST enumeration member 13

resFE_INVALID_ARGUMENT enumeration member 13

resFE_INVALID_STATE enumeration member 13

resFE_LAST enumeration member 13

resFE_NOT_FOUND enumeration member 13

resFE_NOT_IMPLEMENTED enumeration member 13

resFE_NULL_POINTER enumeration member 13

resFE_OUT_OF_MEMORY enumeration member 13

resFE_TIMEOUT enumeration member 13

resFE_UNEXPECTED enumeration member 13

Resolver 741

resS_OK enumeration member 14

resSI_FALSE enumeration member 14

resSI_FIRST enumeration member 14

resSI_LAST enumeration member 14

resSI_TRUE enumeration member 14

resSW_ASYNC_PROCESSING enumeration member 15

resSW_FIRST enumeration member 15

resSW_LAST enumeration member 15

resSW NOTHING DONE enumeration member 15

resSW_WARNING enumeration member 15

Result code messages table definition macros 49

Result manipulation macros 50

RSA crypto algorithm configuration macros 285

S

SAssertCallStackTraceHandler 63

SAssertCallStackTraceHandler struct 63

SAssertFailHandler 63

SAssertFailHandler struct 63

SAssertFinalBehaviorHandler 63

SAssertFinalBehaviorHandler struct 63

SAssertTraceHandler 63

SAssertTraceHandler struct 63

SecurePrng crypto algorithm configuration macros 285

SecureSeed crypto algorithm configuration macros 286

SEnumService 662

SEnumService struct 662

SEnumUri 662

SEnumUri struct 662

ServicingThread 762

SetSockOpt8021QUserPriority 657

SetSockOpt8021QUserPriority function 657

SetSockOptAllowAnySource 658

SetSockOptAllowAnySource function 658

SetSockOptBlocking 658

SetSockOptBlocking function 658

SetSockOptBroadcast 658

SetSockOptBroadcast function 658

SetSockOptIpv6UnicastHops 659

SetSockOptIpv6UnicastHops function 659	Tracing level activation check macros 54
SetSockOptKeepAliveEnable 659	Tracing macros 52
SetSockOptKeepAliveEnable function 659	Types 84, 425, 521
SetSockOptLinger 659	
SetSockOptLinger function 659	
SetSockOptNagle 660	
SetSockOptNagle function 660	
SetSockOptReceiveBufferSize 660	
SetSockOptReceiveBufferSize function 660	
SetSockOptReuseAddress 661	
SetSockOptReuseAddress function 661	
SetSockOptTos 661	
SetSockOptTos function 661	
SetSockOptTransmitBufferSize 661	
SetSockOptTransmitBufferSize function 661	
SetSockOptUdpChecksum 662	
SetSockOptUdpChecksum function 662	
SHA-1 crypto algorithm configuration macros 287	
SHA-1 MAC crypto algorithm configuration macros 287	
SHA-2 crypto algorithm configuration macros 288	
SHA-2 MAC crypto algorithm configuration macros 288	
SLocallpAddress 663	
SLocallpAddress struct 663	
Standard data types 85	
Startup 789	
STraceNode 64	
STraceNode struct 64	
Structures 62, 157, 511, 662, 740, 762, 791	
szVERSION_FW 318	
szVERSION_FW macro 318	
T	
Templates 64, 158, 515	
Thread scheduling policy and priority definitions 275	
Time 816	
Tls 794	
Tls configuration macros 308	
TO 62	
TOA 39	
TOA macro 39	
TOS 39	
TOS macro 39	
Tracing Internal Buffer configuration 313	
U	
uint16_t 85	
uint16_t type 85	
uint32_t 85	
uint32_t type 85	
uint64_t 85	
uint64_t type 85	
uint8_t 85	
uint8_t type 85	
uJAN_1ST_1900 839	
uJAN_1ST_1900 variable 839	
uJAN_1ST_2000 839	
uJAN_1ST_2000 variable 839	
uMD5_HASH_SIZE_IN_BITS 411	
uMD5_HASH_SIZE_IN_BITS variable 411	
uMD5_HASH_SIZE_IN_BYTES 412	
uMD5_HASH_SIZE_IN_BYTES variable 412	
uMD5_MAC_SIZE_IN_BITS 412	
uMD5_MAC_SIZE_IN_BITS variable 412	
uMD5_MAC_SIZE_IN_BYTES 412	
uMD5_MAC_SIZE_IN_BYTES variable 412	
uMEMORY_BLOCKS_INTERVALS 522	
uMEMORY_BLOCKS_INTERVALS variable 522	
uMS_PER_DAY 839	
uMS_PER_DAY variable 839	
Unaligned data types 93	
UnInstallGenericFaultHandler 36	
UnInstallGenericFaultHandler function 36	
UnregisterECom 423	
UnregisterECom function 423	
UnregisterEventObserver 36	
UnregisterEventObserver function 36	
uSEC_PER_DAY 839	
uSEC_PER_DAY variable 839	
uSHA1_HASH_SIZE_IN_BITS 412	
uSHA1_HASH_SIZE_IN_BITS variable 412	
uSHA1_HASH_SIZE_IN_BYTES 412	
uSHA1_HASH_SIZE_IN_BYTES variable 412	
uSHA256_BLOCK_SIZE 412	

uSHA256_BLOCK_SIZE variable 412
uSHA256_HASH_SIZE_IN_BITS 412
uSHA256_HASH_SIZE_IN_BITS variable 412
uSHA256_HASH_SIZE_IN_BYTES 412
uSHA256_HASH_SIZE_IN_BYTES variable 412
Uuid Generator crypto algorithm configuration macros 289

V

Variables 94, 318, 411, 521, 663, 794, 839

X

Xml 839