



M5T SIP SAFE v4.1 - API Reference

Proprietary & Confidential

Legal Information

Copyright © 2010 Media5 Corporation

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: 15 janvier 2010

Reference number

pkg_SipStack-FOR20080128-API Reference

Table of Contents

Introduction	1
Tracing Nodes	2
g_stSipStackSipCore Variable	2
g_stSipStackSipCoreSvc Variable	2
g_stSipStackSipParser Variable	2
g_stSipStackSipProxy Variable	3
g_stSipStackSipTransaction Variable	3
g_stSipStackSipTransport Variable	3
g_stSipStackSipUserAgent Variable	3
M5T SIP SAFE	5
Config	5
Configuring the SIP Stack with "PreSipStackCfg.h"	5
MXD_POST_SIPSTACKCFG	5
MXD_SIPSTACK_ENABLE_TLS	5
MXD_SEQUENTIAL_TRANSACTION_TABLE	5
MXD_SIPSTACK_ENABLE_SIP_KEEP_ALIVE_SVC_SUPPORT	6
MXD_SIPSTACK_ENABLE_SIP_CONNECTION_BLACKLIST_SVC_SUPPORT	6
MXD_SIPSTACK_ENABLE_SIP_CORE_OUTPUT_CONTROLLING_SVC_SUPPORT	6
MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_SVC_SUPPORT	6
MXD_SIPSTACK_ENABLE_SIP_CONNECTION_REUSE_SVC_SUPPORT	7
MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_CURRENT_SRV_FQDN_RETRIEVAL	7
MXD_SIPSTACK_ENABLE_SIP_OUTBOUND_CONNECTION_SVC_SUPPORT	7
MXD_SIPSTACK_ENABLE_SIP_SERVER_LOCATION_SVC_SUPPORT	7
MXD_SIPSTACK_ENABLE_SIP_SPIRALLING_SVC_SUPPORT	7
MXD_SIPSTACK_ENABLE_SIP_SYMMETRIC_UDP_SVC_SUPPORT	8
MXD_SIPSTACK_ENABLE_SIP_STATELESS_DIGEST_SERVER_AUTH_SVC_SUPPORT	8
MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_STATEFUL_PROXY_SVC_SUPPORT	8
MXD_SIPSTACK_ENABLE_SIP_SESSION_STATEFUL_PROXY_SVC_SUPPORT	8
MXD_SIPSTACK_ENABLE_SIP_STATELESS_PROXY_SVC_SUPPORT	8
MXD_SIPSTACK_ENABLE_SIP_DIGEST_CLIENT_AUTH_SVC_SUPPORT	9
MXD_SIPSTACK_ENABLE_SIP_GENERIC_SVC_SUPPORT	9

MXD_SIPSTACK_ENABLE_SIP_GLARE_SVC_SUPPORT	9
MXD_SIPSTACK_ENABLE_SIP_MWI_SVC_SUPPORT	9
MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT	9
MXD_SIPSTACK_ENABLE_SIP_OPTION_TAGS_SVC_SUPPORT	10
MXD_SIPSTACK_ENABLE_SIP_PRIVACY_SVC_SUPPORT	10
MXD_SIPSTACK_ENABLE_SIP_PUBLISH_SVC_SUPPORT	10
MXD_SIPSTACK_ENABLE_SIP_REDIRECTION_SVC_SUPPORT	10
MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT	11
MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT	11
MXD_SIPSTACK_ENABLE_SIP_REGISTRATION_SVC_SUPPORT	11
MXD_SIPSTACK_ENABLE_SIP_RELIABLE_PROVISIONAL_RESPONSE_SVC_SUPPORT	11
MXD_SIPSTACK_ENABLE_SIP_REPLACE_SVC_SUPPORT	11
MXD_SIPSTACK_ENABLE_SIP_SESSION_SVC_SUPPORT	12
MXD_SIPSTACK_ENABLE_SIP_SESSION_TIMER_SVC_SUPPORT	12
MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT	12
MXD_SIPSTACK_ENABLE_SIP_TRANSFER_SVC_07_SUPPORT	12
MXD_SIPSTACK_ENABLE_SIP_UA_ASSERTED_IDENTITY_SVC_SUPPORT	13
MXD_SIPSTACK_ENABLE_SIP_UPDATE_SVC_SUPPORT	13
MXD_SIPSTACK_ENABLE_SIP_USER_AGENT_SVC_SUPPORT	13
MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_COMPLETION_SVC_SUPPORT	13
MXD_SIPSTACK_ENABLE_SIP_DIVERSION_SVC_SUPPORT	13
MXD_SIPSTACK_ENABLE_SIP_SERVER_MONITOR_SVC_SUPPORT	14
MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET Macro	14
MXD_SIPSTACK_ENABLE_SERVER_INVITE_SAME_BRANCH_ACK	14
MXD_SIPSTACK_IPV6_ENABLE_SUPPORT	14
MXD_SIPSTACK_MAILBOX_URI_SPECIFIC_IMPLEMENTATIONS_ENABLE_SUPPORT	14
MXD_SIPSTACK_ENABLE_SIP_STATISTICS_SVC_SUPPORT	15
MXD_SIPSTACK_ENABLE_SIP_ENUM_SVC_SUPPORT	15
MXD_SIPSTACK_MAX_DUMP_LENGTH Macro	15
MXD_SIPSTACK_ENABLE_DEFAULT_DATA_LOGGER	15
MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR	15
MXD_SIPSTACK_ENABLE_REGINFO_SUPPORT	16
MXD_SIPSTACK_ENABLE_SIPCONTEXT_INSTANCE_TRACKING	16
MXD_SIPSTACK_SIP_TRANSACTION_PEER_ADDRESS_MATCH_BYPASS_ENABLE_SUPPORT	16
MXD_SIPSTACK_ENABLE_SIP_DESTINATION_SELECTION_SVC_SUPPORT	16
MXD_SIPSTACK_ENABLE_SIP_VIA_MANAGEMENT_SVC_SUPPORT	17
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPCONTEXT_INSTANCE_TRACKING	17
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPPACKET_INSTANCE_TRACKING	17
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING	17

MXD_SIPSTACK_ENABLE_DEPRECATED_SIPTRANSACTION_INSTANCE_TRACKING	18
MXD_SIPSTACK_ENABLE_SIP_UA_HELPERS_SUPPORT	18
SipCore	18
Classes	18
ISipClientEventControl Class	19
ISipClientTransaction Class	21
ISipContext Class	23
ISipCoreConfig Class	28
ISipCoreUser Class	69
ISipDialogMatcher Class	71
ISipForkedDialogGrouper Class	73
ISipForkedDialogGrouperMgr Class	74
ISipServerEventControl Class	75
Enumerations	77
SipCoreSvc	77
Classes	78
CSipStatisticsContainer Class	78
ISipConnectionBlacklistMgr Class	83
ISipConnectionBlacklistSvc Class	84
ISipConnectionReuseSvc Class	85
ISipCoreOutputControllingMgr Class	88
ISipCoreOutputControllingSvc Class	90
ISipDestinationSelectionSvc Class	92
ISipDiversionSvc Class	93
ISipEnumRequestHandlerMgr Class	94
ISipEnumSvc Class	95
ISipOptionTagsSvc Class	96
ISipOutboundConnectionMgr Class	100
ISipOutboundConnectionSvc Class	102
ISipPersistentConnectionMgr Class	105
ISipPersistentConnectionSvc Class	107
ISipServerLocationSvc Class	108
ISipStatelessDigestServerAuthSvc Class	110
ISipStatisticsInfo Class	116
ISipStatisticsSvc Class	123
ISipSymmetricUdpSvc Class	124
ISipViaManagementSvc Class	125
Enumerations	126
Types	126

mxt_PFNServerLocationListModifier Type	126
SipParser	127
Classes	127
CAbsoluteUri Class	128
CDate Class	133
CGenericParam Class	140
CGenParamList Class	146
CHeaderList Class	155
CHostPort Class	167
CImUri Class	177
CMailboxUri Class	182
CMessageSummary Class	192
CNameAddr Class	201
CPresUri Class	211
CQuotedString Class	215
CRawHeader Class	218
CReginfo Class	222
CRequestLine Class	234
CSipHeader Class	239
CSipMessageBody Class	298
CSipPacketParser Class	312
CSipStatusLine Class	321
CSipUri Class	325
CStringHelper Class	341
CTelUri Class	351
CToken Class	360
CUriFactory Class	378
IUri Class	380
Enumerations	384
ESipHeaderType Enumeration	384
ESipMethod Enumeration	387
ESipStatusClass Enumeration	388
Functions	388
MxConvertSipHeader Function	389
MxConvertSipMethod Function	389
MxConvertSipMethod Function	390
MxGetDefaultReasonPhrase Function	390
MxGetSipStatusClass Function	391
MxSetDefaultHeaderOrder Function	391

Variables	391
g_aszMETHOD_NAME Variable	391
g_auHeaderOrder Variable	392
Status Codes	392
Uri Parameter	394
Header Parameters	395
IUri Constants	398
RegInfo Constants	399
CSipHeader Constants	401
SipProxy	401
Classes	402
CSipProxyConfig Class	402
ISipSessionStatefulProxyMgr Class	405
ISipSessionStatefulProxySvc Class	414
ISipStatelessProxyMgr Class	420
ISipStatelessProxySvc Class	425
ISipTransactionStatefulProxyMgr Class	428
ISipTransactionStatefulProxySvc Class	435
SipTransport	440
Classes	440
CInstanceTracker Class	440
CSipPacket Class	447
ISipDataLogger Class	458
ISipTlsContextFactory Class	460
ISipTransportObserver Class	467
ISipTransportUser Class	470
SipUserAgent	471
Classes	471
ISipDigestClientAuthMgr Class	472
ISipDigestClientAuthSvc Class	474
ISipGenericMgr Class	479
ISipGenericSvc Class	481
ISipGlareMgr Class	483
ISipGlareSvc Class	484
ISipMwiMgr Class	485
ISipMwiSvc Class	489
ISipNotifierMgr Class	492
ISipNotifierSvc Class	497

ISipOptionTagsMgr Class	504
ISipPrivacyMgr Class	505
ISipPrivacySvc Class	507
ISipPublishMgr Class	513
ISipPublishSvc Class	516
ISipRedirectionMgr Class	522
ISipRedirectionSvc Class	523
ISipRefereeMgr Class	527
ISipRefereeSvc Class	531
ISipReferrerMgr Class	535
ISipReferrerSvc Class	542
ISipRegistrationMgr Class	546
ISipRegistrationSvc Class	548
ISipReliableProvisionalResponseMgr Class	559
ISipReliableProvisionalResponseSvc Class	562
ISipReplacesMgr Class	566
ISipReplacesSvc Class	568
ISipSessionMgr Class	569
ISipSessionSvc Class	578
ISipSessionTimerMgr Class	582
ISipSessionTimerSvc Class	584
ISipSubscriberMgr Class	590
ISipSubscriberSvc Class	595
ISipTransactionCompletionMgr Class	604
ISipTransactionCompletionSvc Class	605
ISipTransferMgr07 Class	607
ISipTransferSvc07 Class	614
ISipUaAssertedIdentityMgr Class	623
ISipUaAssertedIdentitySvc Class	628
ISipUpdateMgr Class	633
ISipUpdateSvc Class	637
ISipUserAgentSvc Class	639
Variables	658
Sip Event Constants	658
Startup	660
Classes	660
CSipStackInitializer Class	660

1 - Introduction

Welcome to the M5T SIP API Reference. This document provides quick and simple references to the usage of both the M5T SIP-UA SAFE and M5T SIP-Proxy SAFE products. It should be used by programmers that already have some knowledge about M5T SIP SAFE. Reading "M5T SIP SAFE v4.1 - Programmer's Guide" is usually a good starting point before reading this API reference.

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

2 - Tracing Nodes

2.1 - SipCore tracing nodes

These are the SipCore tracing nodes.

C++

```
STraceNode g_stSipStackSipCoreCSipStackMonitor;
STraceNode g_stSipStackSipCore;
STraceNode g_stSipStackSipCoreResultIdSipCore;
STraceNode g_stSipStackSipCoreCSipForkedDialogGrouper;
STraceNode g_stSipStackSipCoreCSipContext;
STraceNode g_stSipStackSipCoreCSipContextFeatureECOM;
STraceNode g_stSipStackSipCoreCSipDialogMatcherList;
STraceNode g_stSipStackSipCoreCSipReqCtxConnectionSvc;
STraceNode g_stSipStackSipCoreCSipCoreConfig;
STraceNode g_stSipStackSipCoreCSipCoreConfigFeatureECOM;
STraceNode g_stSipStackSipCoreCSipEntity;
STraceNode g_stSipStackSipCoreCSipReqCtxCoreSvc;
STraceNode g_stSipStackSipCoreCSipCoreEventList;
STraceNode g_stSipStackSipCoreCSipNetworkInterfaceList;
STraceNode g_stSipStackSipCoreCSipRequestContext;
STraceNode g_stSipStackSipCoreCSipRequestContextFeatureECOM;
```

2.2 - SipCoreSvc tracing nodes

These are the SipCoreSvc tracing nodes.

C++

```
STraceNode g_stSipStackSipCoreSvcCSipConnectionBlacklist;
STraceNode g_stSipStackSipCoreSvcCSipConnectionBlacklistFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipConnectionBlacklistSvc;
STraceNode g_stSipStackSipCoreSvcCSipConnectionBlacklistSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipCoreOutputControllingSvc;
STraceNode g_stSipStackSipCoreSvcCSipCoreOutputControllingSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipPersistentConnectionList;
STraceNode g_stSipStackSipCoreSvcCSipPersistentConnectionSvc;
STraceNode g_stSipStackSipCoreSvcCSipPersistentConnectionSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipOutboundSvc;
STraceNode g_stSipStackSipCoreSvcCSipOutboundReqCtxSvc;
STraceNode g_stSipStackSipCoreSvcCSipConnectionReuseSvc;
STraceNode g_stSipStackSipCoreSvcCSipConnectionReuseReqCtxCoreSvc;
STraceNode g_stSipStackSipCoreSvcCSipServerLocationSvc;
STraceNode g_stSipStackSipCoreSvcCSipServerLocationSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipReqCtxServerLocationSvc;
STraceNode g_stSipStackSipCoreSvcCSipDestinationSelectionSvc;
STraceNode g_stSipStackSipCoreSvcCReqCtxSipDestinationSelectionSvc;
STraceNode g_stSipStackSipCoreSvcCSipViaManagementSvc;
STraceNode g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvc;
STraceNode g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipSymmetricUdpSvc;
STraceNode g_stSipStackSipCoreSvcCSipSymmetricUdpSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipStatisticsContainer;
STraceNode g_stSipStackSipCoreSvcCSipStatisticsReqCtxSvc;
STraceNode g_stSipStackSipCoreSvcCSipStatisticsSvc;
STraceNode g_stSipStackSipCoreSvcCSipStatisticsSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipEnumRequestHandler;
STraceNode g_stSipStackSipCoreSvcCSipEnumSvc;
STraceNode g_stSipStackSipCoreSvcCSipEnumSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipServerMonitor;
STraceNode g_stSipStackSipCoreSvcCSipServerMonitorFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipServerMonitorSvc;
STraceNode g_stSipStackSipCoreSvcCSipServerMonitorSvcFeatureECOM;
STraceNode g_stSipStackSipCoreSvcCSipServerLocator;
STraceNode g_stSipStackSipCoreSvcCServerLocator;
STraceNode g_stSipStackSipCoreSvcResultIdSipCoreSvc;
```

2.3 - SipParser tracing nodes

These are the SipParser tracing nodes.

C++

```
STraceNode g_stSipStackSipParser;
STraceNode g_stSipStackSipParserResultIdSipParser;
STraceNode g_stSipStackSipParserCDate;
STraceNode g_stSipStackSipParserCSipHeader;
```

2.4 - SipProxy tracing nodes

These are the SipProxy tracing nodes.

C++

```
STraceNode g_stSipStackSipProxy;
STraceNode g_stSipStackSipProxyCSipProxyConfig;
STraceNode g_stSipStackSipProxyCSipSessionStatefulProxySvc;
STraceNode g_stSipStackSipProxyCSipSessionStatefulProxySvcFeatureECOM;
STraceNode g_stSipStackSipProxyCSipTransactionStatefulProxySvc;
STraceNode g_stSipStackSipProxyCSipTransactionStatefulProxySvcFeatureECOM;
STraceNode g_stSipStackSipProxyCSipProxyHelper;
STraceNode g_stSipStackSipProxyCSipStatelessProxySvc;
STraceNode g_stSipStackSipProxyCSipStatelessProxySvcFeatureECOM;
```

2.5 - SipTransaction tracing nodes

These are the SipTransaction tracing nodes.

C++

```
STraceNode g_stSipStackSipTransaction;
STraceNode g_stSipStackSipTransactionResultIdSipTransaction;
STraceNode g_stSipStackSipTransactionCSipClientInviteTransaction;
STraceNode g_stSipStackSipTransactionCSipServerInviteTransaction;
STraceNode g_stSipStackSipTransactionCSipTransaction;
STraceNode g_stSipStackSipTransactionCSipClientNonInviteTransaction;
STraceNode g_stSipStackSipTransactionCSipServerNonInviteTransaction;
STraceNode g_stSipStackSipTransactionCSipTransactionMgr;
```

2.6 - SipTransport tracing nodes

These are the SipTransport tracing nodes.

C++

```
STraceNode g_stSipStackSipTransport;
STraceNode g_stSipStackSipTransportResultIdSipTransport;
STraceNode g_stSipStackSipTransportCSipAsyncSocketFactoryConfigurationMgr;
STraceNode g_stSipStackSipTransportCSipClientSocket;
STraceNode g_stSipStackSipTransportCSipParserSvc;
STraceNode g_stSipStackSipTransportCSipSpirallingSvc;
STraceNode g_stSipStackSipTransportCSipTransportSvc;
STraceNode g_stSipStackSipTransportCSipConnectionSvc;
STraceNode g_stSipStackSipTransportCSipServerSocket;
STraceNode g_stSipStackSipTransportCSipTlsContextFactory;
STraceNode g_stSipStackSipTransportCSipTransportTools;
STraceNode g_stSipStackSipTransportCSipPacket;
STraceNode g_stSipStackSipTransportCSipTransportMgr;
STraceNode g_stSipStackSipTransportCSocketFactory;
STraceNode g_stSipStackSipTransportCSipDefaultDataLogger;
```

2.7 - SipUserAgent tracing nodes

These are the SipUserAgent tracing nodes.

C++

```
STraceNode g_stSipStackSipUserAgentCSipDigestClientAuthSvc;
STraceNode g_stSipStackSipUserAgentCSipDigestClientAuthSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipDiversionSvc;
STraceNode g_stSipStackSipUserAgentCSipDiversionSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipGenericSvc;
STraceNode g_stSipStackSipUserAgentCSipGenericSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvc;
```

```
STraceNode g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipGlareSvc;
STraceNode g_stSipStackSipUserAgentCSipGlareSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipMwiSvc;
STraceNode g_stSipStackSipUserAgentCSipMwiSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipOptionTagsSvc;
STraceNode g_stSipStackSipUserAgentCSipOptionTagsSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipPrivacySvc;
STraceNode g_stSipStackSipUserAgentCSipPrivacySvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipPublishSvc;
STraceNode g_stSipStackSipUserAgentCSipPublishSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipRedirectionSvc;
STraceNode g_stSipStackSipUserAgentCSipRedirectionSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipRefereeSvc;
STraceNode g_stSipStackSipUserAgentCSipRefereeSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipReferrerSvc;
STraceNode g_stSipStackSipUserAgentCSipReferrerSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipRegistrationSvc;
STraceNode g_stSipStackSipUserAgentCSipRegistrationSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvc;
STraceNode g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipReplacesSvc;
STraceNode g_stSipStackSipUserAgentCSipReplacesSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSessionSvc;
STraceNode g_stSipStackSipUserAgentCSipSessionSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSessionTransaction;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUacBye;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUacByeFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUacInvite;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUacInviteFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUasBye;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUasByeFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUasInvite;
STraceNode g_stSipStackSipUserAgentCSipSessionTransactionUasInviteFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipAutomaticAnswererReqCtxSvc;
STraceNode g_stSipStackSipUserAgentCSipSessionTimerSvc;
STraceNode g_stSipStackSipUserAgentCSipSessionTimerSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipTransferSvc07;
STraceNode g_stSipStackSipUserAgentCSipTransferSvc07FeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipUaAssertedIdentitySvc;
STraceNode g_stSipStackSipUserAgentCSipUaAssertedIdentitySvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipUpdateSvc;
STraceNode g_stSipStackSipUserAgentCSipUpdateSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipUserAgentSvc;
STraceNode g_stSipStackSipUserAgentCSipUserAgentSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipNotifierSvc;
STraceNode g_stSipStackSipUserAgentCSipNotifierSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipSubscriberSvc;
STraceNode g_stSipStackSipUserAgentCSipSubscriberSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipTransactionCompletionSvc;
STraceNode g_stSipStackSipUserAgentCSipTransactionCompletionReqCtxConSvc;
STraceNode g_stSipStackSipUserAgentCSipTransactionCompletionSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentCSipUaHelpers;
STraceNode g_stSipStackSipUserAgentCSipContextTerminator;
STraceNode g_stSipStackSipUserAgent;
STraceNode g_stSipStackSipUserAgentCSipClientSvc;
STraceNode g_stSipStackSipUserAgentCSipClientSvcFeatureECOM;
STraceNode g_stSipStackSipUserAgentResultIdSipUserAgent;
```

3 - M5T SIP SAFE

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

3.1 - Config

This section documents the Sources/Config folder of the M5T SIP SAFE stack. It is divided in functional subsections:

3.1.1 - Configuring the SIP Stack with "PreSipStackCfg.h"

The SIP stack comes with the file "Config/SipStackCfg.h", which defines many compilation configuration options and values used throughout the source code. Generally, these values need updating for the specific application being developed with the SIP stack.

To update these default values, you must create the "PreSipStackCfg.h" file with the updated configuration options for your application. "PreSipStackCfg.h" is always included first by "SipStackCfg.h" to retrieve application specific configurations, and then the default configuration options found in "Config/SipStackCfg.h" are applied for all items that have not been configured by the application.

"PreSipStackCfg.h" is not packaged with the SIP stack and must be created for the specific application being developed. This file must be placed somewhere in the compiler search path to permit the retrieval of the application specific configuration options by the SIP stack.

Macros

Macro	Description
MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET (see page 14)	Source Code: #define MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET 20
MXD_SIPSTACK_MAX_DUMP_LENGTH (see page 15)	Source Code: #define MXD_SIPSTACK_MAX_DUMP_LENGTH

3.1.1.1 - MXD_POST_SIPSTACKCFG

Source Code: #define MXD_POST_SIPSTACKCFG

Enables the inclusion of "PostSipStackCfg.h" right at the end of SipStackCfg.h. "PostSipStackCfg.h" is an application provided file that can contain additional configuration options to possibly override the configuration found in PreSipStackCfg.h (see page 5) and SipStackCfg.h.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.2 - MXD_SIPSTACK_ENABLE_TLS

Source Code: #define MXD_SIPSTACK_ENABLE_TLS

Enables the inclusion of the TLS transport into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.3 - MXD_SEQUENTIAL_TRANSACTION_TABLE

Source Code: #define MXD_SEQUENTIAL_TRANSACTION_TABLE

The MXD_SEQUENTIAL_TRANSACTION_TABLE macro makes the internal transaction table use a CVector instead of a CMap. This is allowed for memory footprint reasons. Indeed, enabling this macro and thus using a CVector will lead to slightly less memory usage. Care must be taken however because it will also cause loss of performance, especially if the number of simultaneous transactions is large.

In general, applications should not use this macro unless memory constraints are very severe. However, if the number of simultaneous transactions is large (above 10, for instance), loss of performance will occur that may be unacceptable.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.4 - MXD_SIPSTACK_ENABLE_SIP_KEEP_ALIVE_SVC_SUPPORT

Source Code: `#define MXD_SIPSTACK_ENABLE_SIP_KEEP_ALIVE_SVC_SUPPORT`

Enables the inclusion of the CSipKeepAliveSvc into the build.

Enables the keep alive service which is available in the SipTransport layer. This define does not activate the keep alive service, it simply enables the code that permits its use. The sip-outbound keep alive mechanisms follow the draft-ietf-outbound-15.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.5 - MXD_SIPSTACK_ENABLE_SIP_CONNECTION_BLACKLIST_SVC_SUPPORT

Source Code: `#define MXD_SIPSTACK_ENABLE_SIP_CONNECTION_BLACKLIST_SVC_SUPPORT`

Enables the inclusion of the ISipConnectionBlacklistSvc (see page 84) into the build.

Note that the ISipServerLocationSvc (see page 108) must be enabled to enable this service because this service uses the ISipServerLocationSvc (see page 108).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipConnectionBlacklistSvc (see page 84), MXD_SIPSTACK_ENABLE_SIP_SERVER_LOCATION_SVC_SUPPORT (see page 7), PreSipStackCfg.h (see page 5)

3.1.1.6 - MXD_SIPSTACK_ENABLE_SIP_CORE_OUTPUT_CONTROLLING_SVC_SUPPORT

Source Code: `#define MXD_SIPSTACK_ENABLE_SIP_CORE_OUTPUT_CONTROLLING_SVC_SUPPORT`

Enables the inclusion of the ISipCoreOutputControllingSvc (see page 90) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipCoreOutputControllingSvc (see page 90), PreSipStackCfg.h (see page 5)

3.1.1.7 - MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_SVC_SUPPORT

Source Code: `#define MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_SVC_SUPPORT`

Enables the inclusion of the ISipPersistentConnectionSvc (see page 107) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipPersistentConnectionSvc (see page 107), PreSipStackCfg.h (see page 5)

3.1.1.8 - MXD_SIPSTACK_ENABLE_SIP_CONNECTION_REUSE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_CONNECTION_REUSE_SVC_SUPPORT

Enables the inclusion of the ISipConnectionReuseSvc (see page 85) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipConnectionReuseSvc (see page 85), PreSipStackCfg.h (see page 5)

3.1.1.9 - MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_CURRENT_SRV_FQDN_RETRIEVAL

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_CURRENT_SRV_FQDN_RETRIEVAL

Enables the retrieval of the current SRV target for a given persistent connection. This enables the application to set a callback inside the PersistentConnectionList to get the current SRV FQDN and peer address used to connect.

For example, this new information could be used during the TLS handshaking process to match against a certificate alternate name if the said TLS certificate included SRV FQDN alternate names.

In order to match the retrieved SRV and the TLS handshaking callback, the TLS callback now returns the peer address in order to match it to the SRV callback peer address.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CSipPersistentConnectionList, PreSipStackCfg.h (see page 5)

3.1.1.10 - MXD_SIPSTACK_ENABLE_SIP_OUTBOUND_CONNECTION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_OUTBOUND_CONNECTION_SVC_SUPPORT

Enables the inclusion of the ISipOutboundConnectionSvc (see page 102) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipOutboundConnectionSvc (see page 102), PreSipStackCfg.h (see page 5)

3.1.1.11 - MXD_SIPSTACK_ENABLE_SIP_SERVER_LOCATION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SERVER_LOCATION_SVC_SUPPORT

Enables the inclusion of the ISipServerLocationSvc (see page 108) into the build.

Note that when this service is not enabled, the ISipConnectionBlacklistSvc (see page 84) cannot be enabled.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipServerLocationSvc (see page 108), PreSipStackCfg.h (see page 5)

3.1.1.12 - MXD_SIPSTACK_ENABLE_SIP_SPIRALLING_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SPIRALLING_SVC_SUPPORT

Enables the inclusion of the CSipSpirallingSvc into the build.

Enables the Spiralling Service of the SIP stack. This define enables the code that permits to use the Spiralling Service. By default, the Spiralling Service is activated, the application may make a call to ISipCoreConfig::SetSpirallingSvcState (see page 56). See the

ISipCoreConfig (see page 28) documentation for more information.

Notes

The Spiralling Service is useful for an application that implements SIP server services that may need to send SIP packets to themselves, i.e., in loopback.

Location

Define this in PreSipStackCfg.h.

See Also

CSipSpirallingSvc, PreSipStackCfg.h (see page 5)

3.1.1.13 - MXD_SIPSTACK_ENABLE_SIP_SYMMETRIC_UDP_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SYMMETRIC_UDP_SVC_SUPPORT

Enables the inclusion of the ISipSymmetricSvc into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSymmetricSvc, PreSipStackCfg.h (see page 5)

3.1.1.14 - MXD_SIPSTACK_ENABLE_SIP_STATELESS_DIGEST_SERVER_AUTH_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_STATELESS_DIGEST_SERVER_AUTH_SVC_SUPPORT

Enables the inclusion of the ISipStatelessDigestServerAuthSvc (see page 110) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipStatelessDigestServerAuthSvc (see page 110), PreSipStackCfg.h (see page 5)

3.1.1.15 - MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_STATEFUL_PROXY_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_STATEFUL_PROXY_SVC_SUPPORT

Enables the inclusion of the ISipTransactionStatefulProxySvc (see page 435) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipTransactionStatefulProxySvc (see page 435), PreSipStackCfg.h (see page 5)

3.1.1.16 - MXD_SIPSTACK_ENABLE_SIP_SESSION_STATEFUL_PROXY_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SESSION_STATEFUL_PROXY_SVC_SUPPORT

Enables the inclusion of the ISipSessionStatefulProxySvc (see page 414) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSessionStatefulProxySvc (see page 414), PreSipStackCfg.h (see page 5)

3.1.1.17 - MXD_SIPSTACK_ENABLE_SIP_STATELESS_PROXY_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_STATELESS_PROXY_SVC_SUPPORT

Enables the inclusion of the ISipStatelessProxySvc (see page 425) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipStatelessProxySvc (see page 425), PreSipStackCfg.h (see page 5)

3.1.1.18 - MXD_SIPSTACK_ENABLE_SIP_DIGEST_CLIENT_AUTH_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_DIGEST_CLIENT_AUTH_SVC_SUPPORT

Enables the inclusion of the ISipDigestClientAuthSvc (see page 474) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipStatelessProxySvc (see page 425), PreSipStackCfg.h (see page 5)

3.1.1.19 - MXD_SIPSTACK_ENABLE_SIP_GENERIC_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_GENERIC_SVC_SUPPORT

Enables the inclusion of the ISipGenericSvc (see page 481) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipGenericSvc (see page 481), PreSipStackCfg.h (see page 5)

3.1.1.20 - MXD_SIPSTACK_ENABLE_SIP_GLARE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_GLARE_SVC_SUPPORT

Enables the inclusion of the ISipGlareSvc (see page 484) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipGlareSvc (see page 484), PreSipStackCfg.h (see page 5)

3.1.1.21 - MXD_SIPSTACK_ENABLE_SIP_MWI_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_MWI_SVC_SUPPORT

Enables the inclusion of the ISipMwiSvc (see page 489) into the build.

To enable the ISipMwiSvc (see page 489), ISipSubscriberSvc (see page 595) must also be enabled with MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT (see page 12).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipMwiSvc (see page 489), MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT (see page 12), PreSipStackCfg.h (see page 5)

3.1.1.22 - MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT Updated behavior in 4.1.4

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT

Enables the inclusion of the ISipNotifierSvc (see page 497) into the build.

Updated behaviour: The previous ISipNotifierSvc (see page 497) interface has been deprecated and replaced by this one. The interfaces ISipNotifier[PresenceEventName]Svc where PresenceEventName can stand for "MessageSummary", "Reg", etc. are not supported anymore. To support a presence event using the notifier you simply have to call the AddEvent method on the ISipNotifierSvc (see page 497) interface.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSubscriberSvc (see page 595), ISipMwiSvc (see page 489), PreSipStackCfg.h (see page 5)

3.1.1.23 - MXD_SIPSTACK_ENABLE_SIP_OPTION_TAGS_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_OPTION_TAGS_SVC_SUPPORT

Enables the inclusion of the ISipOptionTagsSvc (see page 96) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipOptionTagsSvc (see page 96), PreSipStackCfg.h (see page 5)

3.1.1.24 - MXD_SIPSTACK_ENABLE_SIP_PRIVACY_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_PRIVACY_SVC_SUPPORT

Enables the inclusion of the ISipPrivacySvc (see page 507) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipPrivacySvc (see page 507), PreSipStackCfg.h (see page 5)

3.1.1.25 - MXD_SIPSTACK_ENABLE_SIP_PUBLISH_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_PUBLISH_SVC_SUPPORT

Enables the inclusion of the ISipPublishSvc (see page 516) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipPublishSvc (see page 516), PreSipStackCfg.h (see page 5)

3.1.1.26 - MXD_SIPSTACK_ENABLE_SIP_REDIRECTION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_REDIRECTION_SVC_SUPPORT

Enables the inclusion of the ISipRedirectionSvc (see page 523) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipRedirectionSvc (see page 523), PreSipStackCfg.h (see page 5)

3.1.1.27 - MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT

Enables the inclusion of the ISipRefereeSvc (see page 531) into the build.

This service needs to be enabled to enable the ISipTransferSvc07 (see page 614).

To enable the ISipRefereeSvc (see page 531), ISipNotifierSvc (see page 497) must also be enabled with MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT (see page 9).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipRefereeSvc (see page 531), MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT (see page 9), PreSipStackCfg.h (see page 5)

3.1.1.28 - MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT

Enables the inclusion of the ISipReferrerSvc (see page 542) into the build.

This service needs to be enabled to enable the ISipTransferSvc07 (see page 614).

To enable the ISipReferrerSvc (see page 542), ISipSubscriberSvc (see page 595) must also be enabled with MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT (see page 12).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipReferrerSvc (see page 542), MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT (see page 12), PreSipStackCfg.h (see page 5)

3.1.1.29 - MXD_SIPSTACK_ENABLE_SIP_REGISTRATION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_REGISTRATION_SVC_SUPPORT

Enables the inclusion of the ISipRegistrationSvc (see page 548) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipRegistrationSvc (see page 548), PreSipStackCfg.h (see page 5)

3.1.1.30 - MXD_SIPSTACK_ENABLE_SIP_RELIABLE_PROVISIONAL_RESPONSE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_RELIABLE_PROVISIONAL_RESPONSE_SVC_SUPPORT

Enables the inclusion of the ISipReliableProvisionalResponseSvc (see page 562) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipReliableProvisionalResponseSvc (see page 562), PreSipStackCfg.h (see page 5)

3.1.1.31 - MXD_SIPSTACK_ENABLE_SIP_REPLACE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_REPLACE_SVC_SUPPORT

Enables the inclusion of the ISipReplacesSvc (see page 568) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipReplacesSvc (see page 568), PreSipStackCfg.h (see page 5)

3.1.1.32 - MXD_SIPSTACK_ENABLE_SIP_SESSION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SESSION_SVC_SUPPORT

Enables the inclusion of the ISipSessionSvc (see page 578) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSessionSvc (see page 578), PreSipStackCfg.h (see page 5)

3.1.1.33 - MXD_SIPSTACK_ENABLE_SIP_SESSION_TIMER_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SESSION_TIMER_SVC_SUPPORT

Enables the inclusion of the ISipSessionTimerSvc (see page 584) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSessionTimerSvc (see page 584), PreSipStackCfg.h (see page 5)

3.1.1.34 - MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT Updated behavior in 4.1.4

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT

Enables the inclusion of the ISipSubscriberSvc (see page 595) into the build.

Updated behaviour: The previous ISipSubscriberSvc (see page 595) interface has been deprecated and replaced by this one. The interfaces ISipSubscriber[PresenceEventName]Svc where PresenceEventName can stand for "MessageSummary", "Reg", etc. are not supported anymore. To support a presence event using the subscriber you simply have to call the AddEvent method on the ISipNotifierSvc (see page 497) interface.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipSubscriberSvc (see page 595), ISipMwiSvc (see page 489), PreSipStackCfg.h (see page 5)

3.1.1.35 - MXD_SIPSTACK_ENABLE_SIP_TRANSFER_SVC_07_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_TRANSFER_SVC_07_SUPPORT

Enables the inclusion of the ISipTransferSvc07 (see page 614) into the build.

To enable the ISipTransferSvc07 (see page 614), ISipRefereeSvc (see page 531) and ISipReferrerSvc (see page 542) must also be enabled with MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT (see page 11) and MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT (see page 11).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipTransferSvc07 (see page 614), MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT (see page 11), MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT (see page 11), PreSipStackCfg.h (see page 5)

3.1.1.36 - MXD_SIPSTACK_ENABLE_SIP_UA_ASSERTED_IDENTITY_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_UA_ASSERTED_IDENTITY_SVC_SUPPORT

Enables the inclusion of the ISipUaAssertedIdentitySvc (see page 628) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipUaAssertedIdentitySvc (see page 628), PreSipStackCfg.h (see page 5)

3.1.1.37 - MXD_SIPSTACK_ENABLE_SIP_UPDATE_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_UPDATE_SVC_SUPPORT

Enables the inclusion of the ISipUpdateSvc (see page 637) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipUpdateSvc (see page 637), PreSipStackCfg.h (see page 5)

3.1.1.38 - MXD_SIPSTACK_ENABLE_SIP_USER_AGENT_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_USER_AGENT_SVC_SUPPORT

Enables the inclusion of the ISipUserAgentSvc (see page 639) into the build. The ISipUserAgentSvc (see page 639) MUST be enabled to use any class from the SipUserAgent folder.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipUserAgentSvc (see page 639), PreSipStackCfg.h (see page 5)

3.1.1.39 - MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_COMPLETION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_COMPLETION_SVC_SUPPORT

Enables the inclusion of the ISipTransactionCompletionSvc (see page 605) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipTransactionCompletionSvc (see page 605), PreSipStackCfg.h (see page 5)

3.1.1.40 - MXD_SIPSTACK_ENABLE_SIP_DIVERSION_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_DIVERSION_SVC_SUPPORT

Enables the inclusion of the ISipDiversionSvc (see page 93) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipDiversionSvc (see page 93), PreSipStackCfg.h (see page 5)

3.1.1.41 - MXD_SIPSTACK_ENABLE_SIP_SERVER_MONITOR_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_SERVER_MONITOR_SVC_SUPPORT

Enables the SIP Server Monitoring feature allowing usage of the ISipServerMonitor and ISipServerMonitorSvc interfaces.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipServerMonitorSvc, ISipServerMonitor, PreSipStackCfg.h (see page 5)

3.1.1.42 - MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET Macro

Source Code: #define MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET 20

C++

```
#define MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET 20
```

Description

Defines the initial capacity of the containers used to hold CSipHeader (see page 239) objects. This value is used to avoid unnecessary memory reallocations while adding headers in SIP packets or during SIP packet parsing.

However, note that SIP packets support holding more headers than this defined value.

Location

Define this in PreSipStackCfg.h or in your makefile to override the default value.

3.1.1.43 - MXD_SIPSTACK_ENABLE_SERVER_INVITE_SAME_BRANCH_ACK

Source Code: #define MXD_SIPSTACK_ENABLE_SERVER_INVITE_SAME_BRANCH_ACK

Enables the proper handling of an ACK request to an INVITE with the same branch value in the topmost via as the INVITE. This behaviour is contrary to RFC 3261, which states that the branch in the topmost via of the ACK request MUST be different than that of the INVITE.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.44 - MXD_SIPSTACK_IPV6_ENABLE_SUPPORT

Source Code: #define MXD_SIPSTACK_IPV6_ENABLE_SUPPORT

Enables the use of IP version 6 in the SIP stack. IPV6 must be enabled in the framework in order to use it in the SIP stack.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipCoreConfig (see page 28), PreSipStackCfg.h (see page 5)

3.1.1.45 - MXD_SIPSTACK_MAILBOX_URI_SPECIFIC_IMPLEMENTATIONS_ENABLE_SUPPORT

Source Code: #define MXD_SIPSTACK_MAILBOX_URI_SPECIFIC_IMPLEMENTATIONS_ENABLE_SUPPORT

Enables the use of specific mailbox URI implementations in the SIP stack.

This option includes the implementation and resolving code for CPresUri (see page 211) and CImUri (see page 177).

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.46 - MXD_SIPSTACK_ENABLE_SIP_STATISTICS_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_STATISTICS_SVC_SUPPORT

Enables the inclusion of the ISipStatisticsSvc (see page 123) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.47 - MXD_SIPSTACK_ENABLE_SIP_ENUM_SVC_SUPPORT

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_ENUM_SVC_SUPPORT

Enables the inclusion of the ISipEnumSvc (see page 95) into the build.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

PreSipStackCfg.h (see page 5)

3.1.1.48 - MXD_SIPSTACK_MAX_DUMP_LENGTH Macro

Source Code: #define MXD_SIPSTACK_MAX_DUMP_LENGTH

C++

```
#define MXD_SIPSTACK_MAX_DUMP_LENGTH 8192
```

Description

The maximum length in bytes of a dump buffer when using CInstanceTracker (see page 440). the default length is 8192.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CInstanceTracker::DebugGetContextDump (see page 443), PreSipStackCfg.h (see page 5)

3.1.1.49 - MXD_SIPSTACK_ENABLE_DEFAULT_DATA_LOGGER

Source Code: #define MXD_SIPSTACK_ENABLE_DEFAULT_DATA_LOGGER

Enables the default data logger, which traces the sent and received SIP packets as a string of ASCII characters.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CSipDefaultDataLogger, PreSipStackCfg.h (see page 5)

3.1.1.50 - MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR *Deprecated*

Source Code: #define MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR

Enables the addition of the "X-M5T-Origin: locally generated" custom header to locally-generated final negative responses (>= 3XX). This header can be used to differentiate a locally-generated response from a remote one.

Global variables `pszX_M5T_ORIGIN_HEADER_NAME` (see page 401) and `pszX_M5T_ORIGIN_HEADER_VALUE` (see page 401) are defined in `CSipHeader.h` to help in handling this custom header.

Notes

This macro is DEPRECATED. Applications MUST now use the new `CSipPacket::IsLocallyGenerated` method.

Location

Define this in `PreSipStackCfg.h` or in your makefile.

See Also

`PreSipStackCfg.h` (see page 5), `CSipHeader.h`, `CSipPacketParser` (see page 312)

3.1.1.51 - MXD_SIPSTACK_ENABLE_REGINFO_SUPPORT New in 4.1.3

Source Code: `#define MXD_SIPSTACK_ENABLE_REGINFO_SUPPORT`

Enables the XML parser for reginfo and GRUU.

Location

Define this in `PreSipStackCfg.h` or in your makefile.

See Also

`PreSipStackCfg.h` (see page 5)

3.1.1.52 - MXD_SIPSTACK_ENABLE_SIPCONTEXT_INSTANCE_TRACKING

Source Code: `#define MXD_SIPSTACK_ENABLE_SIPCONTEXT_INSTANCE_TRACKING`

Enables the tracking of `ISipContext` (see page 23) objects, which may be used to facilitate the debugging of memory leaks. When this macro is enabled, the application may call the following methods: `CInstanceTracker::DebugGetContextTable` (see page 443), `CInstanceTracker::DebugGetNumContextsLeft` (see page 443), `CInstanceTracker::DebugGetContextDump` (see page 443).

Notes

This macro should be enabled only in a debugging process since it activates code that requires considerable CPU processing. Using it may therefore significantly decrease the system performance.

Location

Define this in `PreSipStackCfg.h` or in your makefile.

See Also

`CInstanceTracker::DebugGetContextTable` (see page 443), `CInstanceTracker::DebugGetNumContextsLeft` (see page 443), `CInstanceTracker::DebugGetContextDump` (see page 443), `PreSipStackCfg.h` (see page 5)

3.1.1.53 - MXD_SIPSTACK_SIP_TRANSACTION_PEER_ADDRESS_MATCH_BYPASS_ENABLE_SUPPORT New in 4.1.4

Source Code: `#define MXD_SIPSTACK_SIP_TRANSACTION_PEER_ADDRESS_MATCH_BYPASS_ENABLE_SUPPORT`

Enables the peer IP address matching bypass, so that the peer IP address will not be used to match the packet to an existing transaction.

Location

Define this in `PreSipStackCfg.h` or in your makefile.

See Also

`CSipTransaction`, `PreSipStackCfg.h` (see page 5)

3.1.1.54 - MXD_SIPSTACK_ENABLE_SIP_DESTINATION_SELECTION_SVC_SUPPORT New in 4.1.6

Source Code: `#define MXD_SIPSTACK_ENABLE_SIP_DESTINATION_SELECTION_SVC_SUPPORT`

Enables the destination selection service.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipDestinationSelectionSvc (see page 92), PreSipStackCfg.h (see page 5)

3.1.1.55 - MXD_SIPSTACK_ENABLE_SIP_VIA_MANAGEMENT_SVC_SUPPORT **New in 4.1.6**

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_VIA_MANAGEMENT_SVC_SUPPORT

Enables the Via management service.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipViaManagementSvc (see page 125), PreSipStackCfg.h (see page 5)

3.1.1.56 - MXD_SIPSTACK_ENABLE_DEPRECATED_SIPCONTEXT_INSTANCE_TRACKING **New in 4.1.7**

Source Code: #define MXD_SIPSTACK_ENABLE_DEPRECATED_SIPCONTEXT_INSTANCE_TRACKING

Enables the tracking of ISipContext (see page 23) objects, which may be used to facilitate the debugging of memory leaks. When this macro is enabled, the application may call the following methods: CInstanceTracker::DebugGetContextTable (see page 443), CInstanceTracker::DebugGetNumContextsLeft (see page 443), CInstanceTracker::DebugGetContextDump (see page 443).

Notes

This macro should be enabled only in a debugging process since it activates code that requires considerable CPU processing. Using it may therefore significantly decrease the system performance.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CInstanceTracker::DebugGetContextTable (see page 443), CInstanceTracker::DebugGetNumContextsLeft (see page 443), CInstanceTracker::DebugGetContextDump (see page 443), PreSipStackCfg.h (see page 5)

3.1.1.57 - MXD_SIPSTACK_ENABLE_DEPRECATED_SIPPACKET_INSTANCE_TRACKING **New in 4.1.7**

Source Code: #define MXD_SIPSTACK_ENABLE_DEPRECATED_SIPCONTEXT_INSTANCE_TRACKING (see page 17)

Enables the tracking of CSipPacket (see page 447) objects, which may be used to facilitate the debugging of memory leaks. When this macro is enabled, the application may call the following methods: CInstanceTracker::DebugGetPacketTable (see page 445), CInstanceTracker::DebugGetNumPacketsLeft (see page 444), CInstanceTracker::DebugGetPacketDump (see page 444).

Notes

This macro should be enabled only in a debugging process since it activates code that requires considerable CPU processing. Using it may therefore significantly decrease the system performance.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CInstanceTracker::DebugGetPacketTable (see page 445), CInstanceTracker::DebugGetNumPacketsLeft (see page 444), CInstanceTracker::DebugGetPacketDump (see page 444), PreSipStackCfg.h (see page 5)

3.1.1.58 - MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING **New in 4.1.7**

Source Code: #define MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING

Enables the tracking of ISipRequestContext objects, which may be used to facilitate the debugging of memory leaks. When this macro

is enabled, the application may call the following methods: CInstanceTracker::DebugGetRequestContextTable (see page 446), CInstanceTracker::DebugGetNumRequestContextsLeft (see page 444), CInstanceTracker::DebugGetRequestContextDump (see page 445).

Notes

This macro should be enabled only in a debugging process since it activates code that requires considerable CPU processing. Using it may therefore significantly decrease the system performance.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CInstanceTracker::DebugGetRequestContextTable (see page 446), CInstanceTracker::DebugGetNumRequestContextsLeft (see page 444), CInstanceTracker::DebugGetRequestContextDump (see page 445). PreSipStackCfg.h (see page 5)

3.1.1.59 - MXD_SIPSTACK_ENABLE_DEPRECATED_SIPTRANSACTION_INSTANCE_TRACKING New in 4.1.7

Source Code: #define MXD_SIPSTACK_ENABLE_DEPRECATED_SIPTRANSACTION_INSTANCE_TRACKING

Enables the tracking of CSipTransaction objects, which may be used to facilitate the debugging of memory leaks. When this macro is enabled, the application may call the following methods: CInstanceTracker::DebugGetTransactionTable (see page 446), CInstanceTracker::DebugGetNumTransactionsLeft (see page 444), CInstanceTracker::DebugGetTransactionDump (see page 446).

Notes

This macro should be enabled only in a debugging process since it activates code that requires considerable CPU processing. Using it may therefore significantly decrease the system performance.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

CInstanceTracker::DebugGetTransactionTable (see page 446), CInstanceTracker::DebugGetNumTransactionsLeft (see page 444), CInstanceTracker::DebugGetTransactionDump (see page 446). PreSipStackCfg.h (see page 5)

3.1.1.60 - MXD_SIPSTACK_ENABLE_SIP_UA_HELPERS_SUPPORT New in 4.1.7

Source Code: #define MXD_SIPSTACK_ENABLE_SIP_UA_HELPERS_SUPPORT

Enables helper methods for a user agent.

Location

Define this in PreSipStackCfg.h or in your makefile.

See Also

ISipUaHelpers::Terminate PreSipStackCfg.h (see page 5)

3.2 - SipCore

This section documents the Sources/SipCore folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 18)
- Enumerations (see page 77)

3.2.1 - Classes

This section documents the classes of the Sources/SipCore folder.

Classes

Class	Description
ISipClientEventControl (see page 19)	Interface for managing UAC events, reported upon the reception of a response.
ISipClientTransaction (see page 21)	

ISipContext (see page 23)	
ISipCoreConfig (see page 28)	
ISipCoreUser (see page 69)	
ISipDialogMatcher (see page 71)	
ISipForkedDialogGrouper (see page 73)	
ISipForkedDialogGrouperMgr (see page 74)	
ISipServerEventControl (see page 75)	

3.2.1.1 - ISipClientEventControl Class

Interface for managing UAC events, reported upon the reception of a response.

Class Hierarchy



C++

```
class ISipClientEventControl : public IEComUnknown;
```

Description

This interface is provided as a parameter to all client events reported by stack services. Client events are reported upon the reception of a response to an outgoing request.

It is through this interface that the manager to which the client event is reported can specify how it has handled the event and how it wants the stack to proceed with it.

Refer to the M5T SIP Stack Programmer's Guide for more information on how to use this interface while processing client events.

This is an ECOM interface and as such it is reference counted.

Location

SipCore/ISipClientEventControl.h

See Also

ISipClientTransaction (see page 21)

Methods

Method	Description
• A CallNextClientEvent (see page 19)	Calls the next event in the event list if any.
• A ClearClientEvents (see page 20)	Informs the transaction that the application is not interested in further event processing.
• A GetOpaque (see page 20)	Gets the user-specified opaque parameter associated with this transaction.
• A RelissueRequest (see page 20)	Retries sending the request on a new client transaction.
• A SetOpaque (see page 21)	Sets a user-specified opaque parameter associated with this transaction.

Legend

• A	Method
A	abstract

3.2.1.1.1 - Methods

3.2.1.1.1.1 - ISipClientEventControl::CallNextClientEvent Method

Calls the next event in the event list if any.

C++

```
virtual mxt_result CallNextClientEvent() = 0;
```

Returns

- resS_OK: Next event was successfully called.

- resFE_FAIL: No more event in event list.

Description

Calls the next event in the event list. The manager can use this method when it receives an event and it is not able to take an appropriate action on it. For instance, it could do so when the authentication service ask for credentials for a realm that is unknown to the manager. In this case, CallNextClientEvent would call the failure event on the service that issued the request and the application could then treat the response as a general failure case.

See Also

ClearClientEvent, RelissueRequest (see page 20)

3.2.1.1.1.2 - ISipClientEventControl::ClearClientEvents Method

Informs the transaction that the application is not interested in further event processing.

C++

```
virtual mxt_result ClearClientEvents() = 0;
```

Returns

- resS_OK: The event list has been cleared.

Description

Clears the event list and releases the resources associated with the event and possibly with the transaction.

See Also

RelissueRequest (see page 20), CallNextClientEvent (see page 19)

3.2.1.1.1.3 - ISipClientEventControl::GetOpaque Method

Gets the user-specified opaque parameter associated with this transaction.

C++

```
virtual mxt_opaque GetOpaque() = 0;
```

Returns

The opaque parameter.

Description

Gets the user-specified opaque parameter associated with this transaction.

This is the opaque parameter that is passed to the function that has initiated this client transaction. The same opaque parameter can also be accessed through ISipClientTransaction::GetOpaque (see page 23).

It can be changed after the client transaction is created by using SetOpaque (see page 21) or ISipClientTransaction::SetOpaque (see page 23).

See Also

SetOpaque (see page 21) ISipClientTransaction::GetOpaque (see page 23) ISipClientTransaction::SetOpaque (see page 23)

3.2.1.1.1.4 - ISipClientEventControl::RelissueRequest Method

Retries sending the request on a new client transaction.

C++

```
virtual mxt_result ReIssueRequest(IN mxt_opaque opqTransaction, OUT ISipClientTransaction*& rpNewTransaction) = 0;
```

Parameters

Parameters	Description
OUT ISipClientTransaction*& rpNewTransaction	The new transaction on which the retried request is being sent. Can be NULL when the request cannot be sent.

Returns

- resS_OK: New client transaction is being processed.
- resFE_FAIL: Unable to create new transaction or no final responses were yet received.

Description

Creates a new client transaction and use it to send the request sent on this transaction once again. The request will be sent with the same method, extra headers and content info as the first one but services state should have changed and they will modify the request accordingly. For instance, the redirection service could use a new request-URI or the authentication service could use new credentials.

This method can only be called when a final response has been received on the client transaction. When the manager receives a final response on a transaction, it has to decide whether it wants to use the re-issuing feature for that transaction or not. Good candidates for re-issuing are challenge for username and password known to the application, redirection, session-timer interval too short or any failure response for which the application knows how to (and wants to) change its services state so the request may succeed the next time. The manager cannot use this method if it needs to change the method, the extra headers or the content information of the request in order to make it succeed. If the manager determines that it does not need to call this method (either because it does not know how to modify state to make the request succeed, because it needs to change the method, the extra headers or the content information of the request or simply because the request already succeeded) it must call ClearClientEvent() on this transaction in order to free all resources associated with it.

See Also

ClearClientEvents ([see page 20](#))

3.2.1.1.5 - ISipClientEventControl::SetOpaque Method

Sets a user-specified opaque parameter associated with this transaction.

C++

```
virtual void SetOpaque(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	The opaque value used to set the parameter.

Description

Sets a user-specified opaque parameter associated with this transaction.

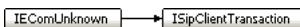
This opaque parameter is the same as the one that can be configured through the ISipClientTransaction ([see page 21](#)) interface. Updating it from one interface impacts the opaque parameter accessible by both interfaces, as they relate to the same transaction.

See Also

GetOpaque ([see page 20](#)) ISipClientTransaction::GetOpaque ([see page 23](#)) ISipClientTransaction::SetOpaque ([see page 23](#))

3.2.1.2 - ISipClientTransaction Class

Class Hierarchy



C++

```
class ISipClientTransaction : public IEComUnknown;
```

Description

This interface represents an ongoing client transaction within the stack.

The application can associate an opaque data with each transaction it creates. This opaque data, which is configured when calling the API that creates and sends a request, is also accessible through this interface.

A client transaction can be cancelled at any time before a final response is received to the request. Simply use the `CancelRequest` (see page 22) API of this interface to cancel a client transaction.

This interface is an ECOM object and as such is reference counted.

The stack user that will never cancel a transaction is not required to keep a reference to a client transaction. For instance, any request other than INVITE requests should never be cancelled. Thus it is possible for the stack user to never keep a reference on the client transaction for all requests, except INVITE requests.

If the stack user did keep a reference to the client transaction, then it must release its reference when the transaction completes, that is when a final response is received.

Refer to the M5T SIP Stack Programmer's Guide for more information on how to use this interface for managing client transactions.

Location

[SipCore/ISipClientTransaction.h](#)

See Also

[ISipClientEventControl](#) (see page 19)

Methods

Method	Description
◆ A <code>CancelRequest</code> (see page 22)	Attempts to cancel client transaction.
◆ A <code>GetOpaque</code> (see page 23)	Gets the user-specified opaque parameter associated with this transaction.
◆ A <code>SetOpaque</code> (see page 23)	Sets a user-specified opaque parameter associated with this transaction.

Legend

◆	Method
A	abstract

3.2.1.2.1 - Methods

3.2.1.2.1.1 - ISipClientTransaction::CancelRequest Method

Attempts to cancel client transaction.

C++

```
virtual mxt_result CancelRequest(IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody) = 0;
```

Parameters

Parameters	Description
IN TO CHeaderList* pExtraHeaders	Additional headers to put in the CANCEL request. It must be NULL to be RFC 3261 compliant. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Message-body information to put in the CANCEL request. It must be NULL to be RFC 3261 compliant. Ownership is TAKEN.

Returns

- `resS_OK`: The Cancel request is being sent or has been buffered until a 1xx class response is received.
- `resFE_FAIL`: Invalid state for sending a CANCEL request.

Description

Attempts to cancel the request sent on the client transaction by creating a CANCEL request corresponding to that original request. The CANCEL is sent as soon as a provisional response to that original request is received. If a provisional response has already been received, the CANCEL is sent immediately.

This method can only be called if no final response has been received for the original transaction.

Since CANCEL is a 'hop by hop' method, additional headers and content information that the application put in this method only reach the next hop. The application is not informed of the status of the CANCEL request. Instead, it is informed of the status of the original response. Note that the original request can still obtain a success response in the case where the success response was sent by the remote party before it received the CANCEL request.

It is not recommended to use CANCEL to cancel requests other than INVITE.

The ISipClientTransaction (see page 21) properly interops with the RFC 2543 style CANCEL, where a final response to the original request is never sent. In this case, the stack automatically generates a final response after a certain amount of time (64*T1).

3.2.1.2.1.2 - ISipClientTransaction::GetOpaque Method

Gets the user-specified opaque parameter associated with this transaction.

C++

```
virtual mxt_opaque GetOpaque() = 0;
```

Returns

The opaque parameter.

Description

Gets the user-specified opaque parameter associated with this transaction.

This is the opaque parameter that is passed to the function that has initiated this client transaction. The same opaque parameter can also be accessed through ISipClientEventControl::GetOpaque (see page 20).

It can be changed after the client transaction is created by using SetOpaque (see page 23) or ISipClientEventControl::SetOpaque (see page 21).

See Also

SetOpaque (see page 23) ISipClientEventControl::GetOpaque (see page 20) ISipClientEventControl::SetOpaque (see page 21)

3.2.1.2.1.3 - ISipClientTransaction::SetOpaque Method

Sets a user-specified opaque parameter associated with this transaction.

C++

```
virtual void SetOpaque(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	The opaque value used to set the parameter.

Description

Sets a user-specified opaque parameter associated with this transaction.

This opaque parameter is the same as the one that can be configured through the ISipClientEventControl (see page 19) interface. Updating it from one interface impacts the opaque parameter accessible by both interfaces, as they relate to the same transaction.

See Also

GetOpaque (see page 23) ISipClientEventControl::GetOpaque (see page 20) ISipClientEventControl::SetOpaque (see page 21)

3.2.1.3 - ISipContext Class

Class Hierarchy



C++

```
class ISipContext : public IEComUnknown;
```

Description

The SIP context is an important concept to grasp to correctly use the stack.

The SIP context defines the scope, features, and capabilities of an application level SIP service. It is only through an `ISipContext` and stack services attached to it that the application can take any SIP actions on the network.

A SIP context is required by a client-based application to offer the following services:

- Making or receiving a SIP call.
- Registering a user.
- Subscribing to a specific event package.
- Starting an IM session

A server-based application would also need a SIP context for the following application level services:

- Proxying one or more received SIP request and its responses.
- Redirecting a received SIP request.
- Receiving a registration request.

A Back-to-Back User-Agent application would also use two SIP contexts, one for each side of the call.

Note that the Server location service MUST always be attached to a context. This is necessary to send a packet with a context even if no name resolution is necessary. This is because that service is also responsible of selecting the right transport protocol and managing reconnection.

The `ISipContext` and the Stack Services

A SIP context alone is not very useful, as it mainly defines a framework upon which different services can be attached to offer various SIP functionality.

By calling `AttachService` (see page 25), the application configures the SIP context with various stack services and, as such, defines the features and scope of the SIP context.

As an example, if an application attaches the SIP User-Agent service to a SIP context, then its scope is that of a User-Agent. The application would also attach additional stack services to the context to enable various functionality on the context, such as providing credentials to a server, managing session timers, and placing a call.

After the application has attached services to a context, it can access them through the standard ECOM mechanism to initiate different SIP actions.

Which action is possible on a SIP context thus depends on the services that have been attached to it. This is also true of the events reported back to the application, as these are generated by the services and not the SIP context.

The lifetime of a SIP context depends on the services that have been attached to it. A SIP context used to place a call is generally good only for the duration of the dialog established by the call, while other SIP contexts, such as a context configured with a stateless proxy service, can be used for as long as the application wants.

Services Ordering

All services attached to a SIP context get to see all packets associated with this context. When doing so, the services may want to report an event to the application, in which case the event is enqueued into an event queue. Once all services have seen the packet, the event queue is processed. For server events (reported upon receiving a request), the stack reports all events from the event queue, one after the other. For client events (reported upon receiving a response to a request), the stack reports only the first event from the event queue.

There is a correlation between the order in which services are attached to a SIP context and the order of generated events. Services attached first get to report their event first to the application. The M5T SIP SAFE Programmer's Guide further discusses this issue and provides general guidance on ordering the services to suit most applications.

The ISipContext is an ECOM object

The ISipContext is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipContext

Interface Id: IID_ISipContext

The ISipContext aggregates all the services that are attached to it. This means that it supports the interfaces (through `QueryIf()`) of all the services that have been attached to it.

Location

SipCore/ISipContext.h

See Also

All services found in the SipUserAgent and SipProxy packages.

Methods

Method	Description
• A AttachService (see page 25)	Attaches a service to the context.
• A Clear (see page 26)	Prepares the context to be deleted.
• A GetEntityId (see page 26)	Gets the current entity ID associated with this SIP context.
• A GetOpaque (see page 26)	Gets current opaque data associated with this SIP context.
• A OnPacketReceived (see page 27)	Tries to handle an incoming packet.
• A SetEntityId (see page 27)	Associates an entity ID with this SIP context.
• A SetOpaque (see page 27)	Associates opaque data to this SIP context.

Legend

• A	Method
A	abstract

3.2.1.3.1 - Methods

3.2.1.3.1.1 - ISipContext::AttachService Method

Attaches a service to the context.

C++

```
virtual mxt_result AttachService(IN mxt_clsid classId) = 0;
```

Parameters

Parameters	Description
IN mxt_clsid classId	The class id of the service to attach to the context.

Returns

resS_OK: The service has been properly added to the context.

resFE_FAIL: The service has not been added either because it failed to be created or it did not implement at least one of the ISipConnectionSvc or the ISipCoreSvc interfaces.

Description

This method is used to attach a service to the SIP context.

After this method returns success, the various interfaces supported by the service can be accessed by calling `QueryIf` on the SIP context.

3.2.1.3.1.2 - ISipContext::Clear Method

Prepares the context to be deleted.

C++

```
virtual mxt_result Clear() = 0;
```

Returns

resSW_SIPCORE_ACTIVE_DIALOG: A dialog was active when this method was called. The application should have terminated the dialog before calling this method. The dialog has been terminated.

resS_OK: No dialog was active when this method was called.

Description

This method must be called before the last reference kept on the context by the application is released. It prepares the context to be deleted.

Normally, this method should be called when no requests or dialogs are pending. However, the method prepares the context to be released even if there are pending requests or dialogs.

After this method is called, the service managers are no longer notified about anything.

If the application still has unused ISipServerEventControl (see page 75) or ISipClientEventControl (see page 19), it MUST use them AND release their reference in order to allow the context to be deleted. In particular, a final response MUST be sent on any ISipServerEventControl (see page 75) related to this ISipContext (see page 23) BEFORE this method is called."

Note that the reference kept on the ISipContext (see page 23) MUST still be released even if this method is called.

See Also

ISipServerEventControl (see page 75), ISipClientEventControl (see page 19)

3.2.1.3.1.3 - ISipContext::GetEntityId Method

Gets the current entity ID associated with this SIP context.

C++

```
virtual unsigned int GetEntityId() = 0;
```

Returns

The entity ID of this SIP context.

Description

Allows the user to access the entity ID associated with this SIP context.

See Also

SetEntityId (see page 27)

3.2.1.3.1.4 - ISipContext::GetOpaque Method

Gets current opaque data associated with this SIP context.

C++

```
virtual mxt_opaque GetOpaque() = 0;
```

Returns

Previously associated opaque data. Opaque with a value of zero if no opaque data had been previously set.

Description

Allows the SIP context user to access the opaque data associated with this SIP context.

See Also

[SetOpaque](#) (see page 27)

3.2.1.3.1.5 - ISipContext::OnPacketReceived Method

Tries to handle an incoming packet.

C++

```
virtual mxt_result OnPacketReceived(IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The packet that was just received.

Returns

resS_OK: The packet has been handled by the context.

resFE_FAIL: The incoming packet has not been handled.

resFE_DUPLICATE: Unable to create the server transaction, as it already exists.

Description

This method is used to have the SIP context and the services it holds process the incoming packet.

The result returned by this method usually depends on the services that are attached to the SIP context.

When calling this on a newly created SIP context that has stateful services attached to it (UA service or stateful proxy service), it is possible that resFE_DUPLICATE is returned, in which case the context should be released immediately. This usually happens when the time between a request and its retransmissions is too short.

3.2.1.3.1.6 - ISipContext::SetEntityId Method

Associates an entity ID with this SIP context.

C++

```
virtual void SetEntityId(IN unsigned int uEntityId) = 0;
```

Parameters

Parameters	Description
IN unsigned int uEntityId	The entity ID.

Description

Allows the user of the SIP context to assign it an entity ID. An entity ID is used to associate a listening port with a SIP context via the given "entity". Packets sent from this SIP context will only use a listening interface with the same entity ID to send the packet. If no listening interface is configured with the same entity ID, the packet will fail to be sent. The VIA and the contact will be automatically updated according to the selected interface. By default, the entity ID is set to 0.

The entity ID should be set immediately after creation of the SipContext and never modified again.

See Also

[GetEntityId](#) (see page 26), [ISipCoreConfig::SetEntityId](#) (see page 48)

3.2.1.3.1.7 - ISipContext::SetOpaque Method

Associates opaque data to this SIP context.

C++

```
virtual void SetOpaque(IN mxt_opaque opaque) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opaque	The opaque data. The stack never uses this data.

Description

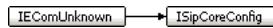
Allows the user of the SIP context to associate data with this context. This is opaque data to the SIP context and is never used or interpreted.

See Also

GetOpaque (see page 26)

3.2.1.4 - ISipCoreConfig Class

Class Hierarchy



C++

```
class ISipCoreConfig : public IEComUnknown;
```

Description

This is the configuration interface of the SIP stack. All dynamic configuration is performed through this interface.

This interface also offers a unified Startup (see page 66) and Shutdown mechanism to make it easy to start and stop the SIP stack.

You can safely create an ISipCoreConfig object whenever it is needed, and release it as soon as the configuration has been updated. The SIP stack suffers no side effect if the application releases all references it has to an ISipCoreConfig.

Listening Address Configuration

RFC 3261 defines three transport protocols for SIP: UDP, TCP and TLS. According to RFC 3261, a device should listen on port 5060 for UDP and TCP, and it should listen on port 5061 for TLS connections.

When advertising that a device listens on default ports in SIP, whether in Contact or Via headers, a device does not have to specify the port it is listening to. The device only has to advertise the port when listening on something else than the default port(s).

The following are some examples:

Contact: sip:user@10.12.13.14	<- Listening on default port(s)
Contact: sip:user@10.12.13.14:5062	<- Listening on non- default port

It is quite important that devices that want to simultaneously support UDP/TCP and TLS listen on the default ports. Not doing so could cause the following problem.

Assume a device listening on 5065 in UDP/TCP and on 5070 for TLS. The user makes a call to `sip:destination@somewhere.com`. Upon resolving, the stack uses TLS to establish the connection to the proxy at `somewhere.com`. The contact in the outgoing INVITE request is:

Contact: sip:user@10.12.13.14:5070

From here, there are different possible scenarios where using this non-default port would break communications:

- If the TLS connection goes down and the proxy tries to re-establish it with UDP or TCP, the device is expecting TLS on 5070.
- If the proxy does not record-route and the peer device does not support TLS, it will try UDP/TCP on 5070, failing again.
- Other scenarios exist when performing transfer and such...

Now if the default ports would have been used instead of non-default ones, then the contact would look like:

Contact: sip:user@10.12.13.14

Under such condition, the device sending a request will use the proper destination port based on the transport protocol being used; 5060 for UDP/TCP and 5061 for TLS.

Initialization Procedure

- Create an **ISipCoreConfig**.
- Create the servicing threads for the SIP stack.
- Configure the SIP stack threads.
- Configure the core user.
- Configure any other optional parameter.
- Call **Startup** (see page 66).
- Call **AddLocalAddress** (see page 33) to add a local network interface.
- Call **ListenA** (see page 41) for all transports and network interfaces to which you need to listen.
- You can safely release the **ISipCoreConfig**.
- You can now start using the SIP stack and its services.

Shutdown Procedure

- Send final responses to all pending server transactions.
- Call **Clear** on all **ISipContexts** on which the SIP stack user still has a reference.
- Release all references to all **ISipContexts** the user still has.
- Call **ShutdownA** (see page 65).
- Wait for **ISipCoreUser::EvShutdownComplete**.

The **ISipCoreConfig** is an ECOM object

The **ISipCoreConfig** is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipCoreConfig

Interface Id: IID_ISipCoreConfig

Location

`SipCore/ISipCoreConfig.h`

Methods

Method	Description
•  AddLocalAddress (see page 33)	Adds a local address.
•  AddServerToResponses (see page 34)	Determines whether or not the Server header is added to responses created by the SIP stack.
•  AddStackVersionTold (see page 35)	Determines whether the SIP stack version is added in the User-Agent and Server headers.
•  AddSupportedExtension (see page 35)	Adds a globally supported extension.
•  AddTransportObserverA (see page 35)	Adds a transport observer for handling specific SIP transport events.
•  AddUserAgentToRequests (see page 36)	Determines whether or not the User-Agent header is added to requests created by the SIP stack.
•  AddViaRportParam (see page 36)	Adds the rport parameter to the Via header
•  CloseAllConnections (see page 37)	Closes all active connections or only those that match the provided remote address.
•  EnableOutboundCrlfKeepAliveA (see page 37)	Enables the CRLF keep alive for reliable transports.
•  ForceVisibleLocalAddress (see page 37)	Allows the application to override the hostport part of the Via and Record-Route headers in sent packets.

• A GetConnectionBlacklistInstance (see page 38)	Allows access to the persistent connection list module.
• A GetNetworkInterfaceList (see page 39)	Gets the network interfaces address list.
• A GetPersistentConnectionList (see page 39)	Allows access to the persistent connection list module.
• A GetSupportedExtensions (see page 39)	Gets the globally supported extension list.
• A GetSupportedIpVersion (see page 40)	This gets the currently configured supported IP version.
• A GetTlsContextFactory (see page 40)	Get the TLS context factory used to create TLS contexts.
• A GetUaHelpers (see page 40)	Allows access to the user agent helpers methods.
• A GetVersion (see page 41)	Obtains a NULL-terminated string representing the version of the SIP stack.
• A ListenA (see page 41)	Listens for incoming SIP packets on an IP address.
• A RemoveLocalAddress (see page 41)	Removes a local address.
• A RemoveSupportedExtension (see page 42)	Removes a globally supported extension.
• A RemoveTransportObserverA (see page 42)	Removes a transport observer from the observer list
• A SetApplicationId (see page 43)	Sets the application identification to output in User-Agent and Server headers.
• A SetClientAuthUnknownQopBehavior (see page 43)	Sets the behavior of the ISipDigestClientAuthSvc (see page 474) upon reception of an unknown qop parameter value in the WWW-Authenticate or Proxy-Authenticate header.
• A SetCommaSeparatedHeader (see page 43)	Sets comma generation setting for the specified header type.
• A SetConnectionBlacklistInstance (see page 44)	Allows setting a new persistent connection list module to use, otherwise the default one is used.
• A SetConnectionParameters (see page 44)	Configures SIP stack connection parameters.
• A SetCoreThread (see page 46)	Sets the thread used by the SIP core and transactions modules.
• A SetCoreUser (see page 47)	Configures the core user interface, which receives new packets and chooses how they are handled.
• A SetCSeq64BitsSupport (see page 47)	Sets if a 64 bits sequence number is supported.
• A SetDefaultDialogMatcherList (see page 47)	Sets the default dialog matcher list.
• A SetDnsResolverThread (see page 48)	Sets the thread that is used to serially process DNS requests.
• A SetEntityId (see page 48)	
• A SetEnumE164ZoneSuffix (see page 49)	Sets the E164 zone suffix for ENUM requests.
• A SetFailoverMode (see page 49)	Sets the failover mode.
• A SetHandshakeValidatorCallback (see page 50)	Configures a callback for the handshake validation.
• A SetHeaderFormPreference (see page 51)	Configures how SIP headers are generated.
• A SetKeepAliveExtensionMgrA (see page 52)	Sets a keep alive extension manager.
• A SetMaxForwards (see page 52)	Sets the value to put in the Max-Forwards header of the requests created by this class.
• A SetMaxPayloadSize (see page 52)	Sets the maximum payload size that can be accepted by the SIP stack.
• A SetMaxReceivePacketSize (see page 53)	Sets the maximum receivable size of SIP packet that can be accepted by the SIP stack.
• A SetMaxSendBufferSize (see page 53)	Sets the socket's transmission buffer size.
• A SetPacketInspectorCallback (see page 54)	Configures a callback that the SIP stack calls upon the reception of a complete SIP packet.
• A SetPacketModifierCallback (see page 54)	Configures a callback that the SIP stack calls upon the reception and parsing of a complete SIP packet.
• A SetPrincipalBufferSize (see page 55)	Sets the size of the principal buffer.
• A SetSipDataLogger (see page 55)	Configures the data logger.
• A SetSocketClosureType (see page 56)	
• A SetSpirallingSvcState (see page 56)	Sets the Spiralling Service state.
• A SetSupportedDnsQueries (see page 56)	Sets the type(s) of DNS queries supported by the application.
• A SetSupportedIpVersion (see page 57)	Configures which versions of TCP/IP are supported or enabled by the application.
• A SetSupportedSipTransport (see page 58)	Sets the transport protocol the SIP stack may use for sending SIP packets.
• A SetT1 (see page 58)	Overrides the value to use for T1.
• A SetT2 (see page 59)	Overrides the value to use for T2.
• A SetT4 (see page 59)	Overrides the value to use for T4.
• A SetTimeoutTimer (see page 59)	Sets the timeout value used by some SIP timers and timeouts.
• A SetTimerB (see page 60)	Overrides the value to use for the Timer B (as specified in RFC 3261).
• A SetTimerD (see page 61)	Overrides the value to use for the Timer D (as specified in RFC 3261).
• A SetTimerF (see page 61)	Overrides the value to use for the Timer F (as specified in RFC 3261).
• A SetTimerH (see page 62)	Overrides the value to use for the Timer H (as specified in RFC 3261).

• A SetTimerJ (see page 62)	Overrides the value to use for the Timer J (as specified in RFC 3261).
• A SetTlsSessionCacheMaxSize (see page 63)	Sets the maximum size of the cached TLS session list.
• A SetTransportThread (see page 63)	Sets the thread that is used by the SIP transport module and the sockets it uses.
• A SetUaResponseMultipleViasCheck (see page 64)	Sets the multiple Via headers check in incoming responses destined to user-agents.
• A SetUdpMaxSizeThreshold (see page 65)	Sets the threshold above which a request is first tried using TCP when no transport mechanism is specified (as per RFC 3261 section 18.1.1).
• A SetViaAddressType (see page 65)	Configures which type of addresses the SIP stack uses in Via headers.
• A ShutdownA (see page 65)	Shuts down the SIP stack asynchronously.
• A Startup (see page 66)	Starts the initialization procedures of the SIP stack.
• A StopListeningA (see page 66)	Stops listening for incoming SIP packets on an IP address.
• A UpdateParserGrammar (see page 67)	Updates the grammar of a given character set.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
EAddressTypePreference (see page 67)	Enumerations describing the preference for how addresses are represented.
ECommaSeparatedHeaderConfig (see page 67)	Configuration options for SetCommaSeparatedHeader (see page 43).
EFailoverMode (see page 68)	
EHeaderFormPreference (see page 68)	Configuration options for SetHeaderFormPreference (see page 51).
ElpVersionConfig (see page 68)	Enumeration defining the possible combination of supported versions of IP.
ESpirallingSvcState (see page 69)	Enumeration defining the Spiralling Service state.

Structs

Struct	Description
SAccessibleNetwork (see page 31)	Struct describing an accessible network.
SNetworkIf (see page 32)	Struct describing a local network interface address and its respective listening port. Refer to GetNetworkInterfaceList (see page 39)() for more information.

3.2.1.4.1 - Structs

3.2.1.4.1.1 - ISipCoreConfig::SAccessibleNetwork Struct

Struct describing an accessible network.

C++

```
struct SAccessibleNetwork {
    CSocketAddr m_address;
    CSocketAddr m_netmask;
};
```

Description

Configures m_address to hold the address of the network, and m_netmask to hold the network mask to apply to a destination address. After the mask has been applied on the destination address, if it equals m_address, then the local address associated with this network is used to send the packet.

Example 1

Assume a device with an interface to 10.1.x.x, it could then configure m_address to 10.1.0.0 and m_mask to 255.255.0.0.

If the SIP stack has to send a packet to 10.1.2.3, the destination address would be masked with m_netmask, resulting in 10.1.0.0. Since the masked destination address and m_address match, the address for which this SAccessibleNetwork is configured would be used.

Example 2

Assume a device that can reach the following networks from a single address, let's say 10.2.5.100:

- 10.0.x.x

- 10.1.x.x
- 10.2.x.x
- 10.3.x.x
- 10.4.x.x
- 10.5.x.x
- 10.6.x.x
- 10.7.x.x

There are multiple ways that the application could configure its list of accessible networks. One way would be to associate with its 10.2.5.100 local address multiple accessible networks:

- 10.0 network: m_address: 10.0.0.0, m_netmask: 255.255.0.0
- 10.1 network: m_address: 10.1.0.0, m_netmask: 255.255.0.0
- 10.2 network: m_address: 10.2.0.0, m_netmask: 255.255.0.0
- 10.3 network: m_address: 10.3.0.0, m_netmask: 255.255.0.0
- 10.4 network: m_address: 10.4.0.0, m_netmask: 255.255.0.0
- 10.5 network: m_address: 10.5.0.0, m_netmask: 255.255.0.0
- 10.6 network: m_address: 10.6.0.0, m_netmask: 255.255.0.0
- 10.7 network: m_address: 10.7.0.0, m_netmask: 255.255.0.0

Another way would be to tweak m_netmask and add a single entry for all networks:

- 10.[0-7] network: m_address: 10.0.0.0, m_netmask: 255.248.0.0

When sending to a destination that is part of one of the network, let's say 10.4.3.65, then the destination is masked with m_netmask. This results in 10.0.0.0. The masked destination matches m_address, thus the associated local address would be used to reach this destination.

When sending to a destination that is not part of one of the network, let's say 10.9.22.98, then the destination is masked with m_netmask. This results in 10.8.0.0. The masked destination does not match m_address, thus the associated local address would not be used to reach this destination.

Notice that the mask is applied in binary. Thus for this example, if representing only the important part of the address in binary:

Destination	Dest. Binary	Mask	Masked Result
10.4.3.65	10.b00000100.3.65	255.b11111000.0.0	10.0.0.0
10.9.22.98	10.b00001001.22.98	255.b11111000.0.0	10.b00001000.0.0

Members

Members	Description
CSocketAddr m_address;	The address of the network.
CSocketAddr m_netmask;	The address mask to apply to destination addresses.

3.2.1.4.1.2 - ISipCoreConfig::SNetworkIf Struct

Struct describing a local network interface address and its respective listening port. Refer to GetNetworkInterfaceList (see page 39)() for more information.

Class Hierarchy

C++

```
struct SNetworkIf {
    CSocketAddr m_socketAddr;
    uint16_t m_uListeningUdpTcpPort;
    uint16_t m_uListeningTlsPort;
};
```

Constructors

Constructor	Description
SNetworkIf (see page 33)	Constructor.

Legend

	Constructor
	Method

3.2.1.4.1.2.1.1 - Data Members

3.2.1.4.1.2.1.1 - ISipCoreConfig::SNetworkIf::m_socketAddr Data Member

Local network interface address.

C++

```
CSocketAddr m_socketAddr;
```

3.2.1.4.1.2.1.2 - ISipCoreConfig::SNetworkIf::m_uListeningTlsPort Data Member

TLS listened port.

C++

```
uint16_t m_uListeningTlsPort;
```

Description

If 0 = none.

3.2.1.4.1.2.1.3 - ISipCoreConfig::SNetworkIf::m_uListeningUdpTcpPort Data Member

UDP or TCP listened port.

C++

```
uint16_t m_uListeningUdpTcpPort;
```

Description

If 0 = none.

3.2.1.4.1.2.2 - Constructors

3.2.1.4.1.2.2.1 - ISipCoreConfig::SNetworkIf::SNetworkIf Constructor

Constructor.

C++

```
SNetworkIf();
```

3.2.1.4.2 - Methods

3.2.1.4.2.1 - ISipCoreConfig::AddLocalAddress Method

Adds a local address.

C++

```
virtual mxt_result AddLocalAddress(IN const CSocketAddr& rAddr, IN TO const CVector<CString>* pvecStrFqdn, IN TO const CVector<SAccessibleNetwork>* pvecDestinations, OUT mxt_opaque& ropqAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddr	The IP address to configure.
IN TO const CVector<CString>* pvecStrFqdn	The FQDNs associated with this local address. It can be NULL, in which case the SIP stack assumes there are no FQDNs associated with this address. Ownership is taken.
IN TO const CVector<SAccessibleNetwork>* pvecDestinations	The accessible networks from this local IP address. It can be NULL, in which case the SIP stack assumes all networks are reachable from this address. Ownership is taken.
OUT mxt_opaque& ropqAddress	SIP Stack-specific opaque address identifier. This can be used by the application to later perform additional operations from this local address.

Returns

resFE_INVALID_STATE: The provided IP address is already present.
 resFE_INVALID_ARGUMENT: The provided IP address is invalid.
 resS_OK: The IP address has been added successfully.

Description

This method configures a local address and its parameters. The SIP stack does not listen to this local address until the user calls ListenA (see page 41).
 An application that has a single address should not configure any accessible network. That is, pvDestinations should be NULL.
 An application that has multiple addresses to different networks should add the addresses in order of preference, as the SIP stack searches a local address to use in order they were added in the SIP stack.
 The last address added should be the default address to use and should have a NULL pvDestinations. This means that there is at least a default local address to use when sending a packet to an unconfigured accessible network.

When pvecStrFqdn contains more than one FQDN and the SIP stack is configured to use FQDNs in Via headers it created, the first FQDN of this list is used.

See Also

RemoveLocalAddress (see page 41), SAccessibleNetwork (see page 31), ListenA (see page 41), StopListeningA (see page 66), SetViaAddressType (see page 65)

Example

An application has two local addresses: 10.2.5.100 to access a private network and 205.237.248.241 to access the public Internet.
 From its 10.2.5.100 address, it has access to 8 networks, from 10.0.x.x to 10.7.x.x.
 From its 205.237.248.241 address, it has access to the public network.

The application would then configure its 10.2.5.100 as its first address with a single SAccessibleNetwork (see page 31) that has the following

3.2.1.4.2.2 - ISipCoreConfig::AddServerToResponses Method

Determines whether or not the Server header is added to responses created by the SIP stack.

C++

```
virtual void AddServerToResponses( IN bool bAdd ) = 0;
```

Parameters

Parameters	Description
IN bool bAdd	The Server header is added to responses when true, otherwise it is not added.

Description

Configures whether or not to add the Server header to responses created by the SIP stack. To determine what to put in that header, refer to SetApplicationId (see page 43) and AddStackVersionTold (see page 35). Note that if added, the header is added only to responses created by the SIP stack and not to responses forwarded by it.

By default, the Server header is added to responses.

This method can be called at any time.

See Also

[SetApplicationId](#) (see page 43), [AddStackVersionTold](#) (see page 35), [AddServerToResponses](#)

3.2.1.4.2.3 - **ISipCoreConfig::AddStackVersionTold** Method

Determines whether the SIP stack version is added in the User-Agent and Server headers.

C++

```
virtual void AddStackVersionToId(IN bool bAdd) = 0;
```

Parameters

Parameters	Description
IN bool bAdd	The SIP stack version is added when true, otherwise it is not added.

Description

Configures whether the SIP stack version should be added to the User-Agent and Server headers. By default, it is added in the form "M5T SIP Stack/major.minor.release.build" at the end of the headers if they are added. 'major', 'minor', 'release' and 'build' are obviously replaced by the corresponding numbers.

Note that this option has no impact on whether or not the User-Agent and Server headers are added to packets sent by the SIP stack.

Note that the application identification set through [SetApplicationId](#) (see page 43) appears in both the User-Agent and Server headers, before the SIP stack version.

If the bAdd parameter is set to false, the SIP stack version is not in the User-Agent and Server headers.

By default, the SIP stack version is added to the User-Agent and Server headers.

This method can be called at any time.

See Also

[SetApplicationId](#) (see page 43), [AddUserAgentToRequests](#) (see page 36), [AddServerToResponses](#) (see page 34)

3.2.1.4.2.4 - **ISipCoreConfig::AddSupportedExtension** Method

Adds a globally supported extension.

C++

```
virtual mxt_result AddSupportedExtension(IN const CString& rstrExtension) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrExtension	The custom extension to add.

Returns

resS_OK: The extension was added. resFE_FAIL: The extension was already present.

Description

Adds a globally supported extension to the list.

3.2.1.4.2.5 - **ISipCoreConfig::AddTransportObserverA** Method

Adds a transport observer for handling specific SIP transport events.

C++

```
virtual mxt_result AddTransportObserverA(IN ISipTransportObserver* pObserver, IN ISipTransportObserver::EInsertObserverPriority ePriority = ISipTransportObserver::eLOW_PRIORITY_OBSERVER) = 0;
```

Parameters

Parameters	Description
IN ISipTransportObserver* pObserver	Pointer to the observer to add.
IN ISipTransportObserver::EInsertObserverPriority ePriority = ISipTransportObserver::eLOW_PRIORITY_OBSERVER	Observer priority. eLOW_PRIORITY_OBSERVER is default.

Returns

resS_OK: The transport observer is correctly added.
 resFE_INVALID_STATE: The SIP stack is not properly started.

Description

This method is used to append a new transport observer of activities on sockets.
 eHIGH_PRIORITY_OBSERVER observers handle transport events before eLOW_PRIORITY_OBSERVER observers.
 When two transport observers are added with the same priority, the last added observer handles transport events first.

See Also

ISipTransportObserver (see page 467), RemoveTransportObserverA (see page 42)

3.2.1.4.2.6 - ISipCoreConfig::AddUserAgentToRequests Method

Determines whether or not the User-Agent header is added to requests created by the SIP stack.

C++

```
virtual void AddUserAgentToRequests(IN bool bAdd) = 0;
```

Parameters

Parameters	Description
IN bool bAdd	The User-Agent header is added to requests when true, otherwise it is not added.

Description

Configures whether or not to add the User-Agent header to requests created by the SIP stack. To determine what to put in that header, refer to SetApplicationId (see page 43) and AddStackVersionTold (see page 35). Note that if added, the header is added only to requests created by the SIP stack and not to requests forwarded by it.

By default, the User-Agent header is added to requests.

This method can be called at any time.

See Also

SetApplicationId (see page 43), AddStackVersionTold (see page 35), AddServerToResponses (see page 34)

3.2.1.4.2.7 - ISipCoreConfig::AddViaRportParam Method

Adds the rport parameter to the Via header

C++

```
virtual void AddViaRportParam(IN bool bAddRport) = 0;
```

Parameters

Parameters	Description
IN bool bAddRport	True to add an rport parameter to the Via header in all requests, false otherwise.

Description

Adds the rport parameter to the Via header of all outgoing requests. RFC 3581 permits the addition of the rport parameter to force UAs to reply on the same UDP address/port on which the request was received. The local address/port used to receive the packet should also be used for NAT traversal purposes.

3.2.1.4.2.8 - ISipCoreConfig::CloseAllConnections Method

Closes all active connections or only those that match the provided remote address.

C++

```
virtual mxt_result CloseAllConnections( IN CSocketAddr* pRemoteAddress = NULL ) = 0;
```

Parameters

Parameters	Description
IN CSocketAddr* pRemoteAddress = NULL	Specify the remote address of the connections to close. Provide NULL to close all active connections.

Returns

- resS_OK: Connections have been successfully closed.
- resFE_FAIL: The operation has not been completed.

Description

Closes all active connections or only the connections that match the pRemoteAddr parameter when it is non-NULL.

This method only closes sockets that are connected. This is true for all transports. This means that listening sockets and sockets used by the symmetric UDP service are left untouched. It is the application's responsibility to close those sockets by the appropriate mechanism.

The sockets are closed by using the closure type set with the ISipCoreConfig::SetSocketClosureType (see page 56) method.

See Also

SetSocketClosureType (see page 56)

3.2.1.4.2.9 - ISipCoreConfig::EnableOutboundCrlfKeepAliveA Method New in 4.1.6

Enables the CRLF keep alive for reliable transports.

C++

```
virtual mxt_result EnableOutboundCrlfKeepAliveA( IN bool bEnable ) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	True to enable the CRLF keep alive. False indicates that there will be no keep alives using CRLF.
Return	resFE_INVALID_STATE: the transport manager is not set.
ress_OK	otherwise.

Description

Enables the CRLF keep alive mechanism for reliable transports as described in the draft-ietf-sip-outbound-15.

By default, this keep alive mechanism is enabled when the compiling switch MXD_SIPSTACK_ENABLE_SIP_KEEP_ALIVE_SVC_SUPPORT (see page 6) is activated. The application needs to enable the mechanism in order to achieve sending keep alives to the server on reliable connections such as TCP and TLS. When enabled, the client sends keep alives using two CRLFs on each flow selected by the application.

Usually the application starts issuing keep alives when it receives a response to a REGISTER request that contains the Require header with outbound value.

When the CRLF keep alive is enabled, the stack is responsible to send the keep alives and receive the responses. The application does not need to interact with the stack for the CRLF keep alives.

3.2.1.4.2.10 - ISipCoreConfig::ForceVisibleLocalAddress Method new in v4.1.8

Allows the application to override the hostport part of the Via and Record-Route headers in sent packets.

C++

```
virtual mxt_result ForceVisibleLocalAddress(IN mxt_opaque opqAddress, IN TOA CHostPort* pVisibleAddress) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqAddress	The opaque that identifies the IP address to be overridden. This opaque has been received from ISipCoreConfig::AddLocalAddress (see page 33).
IN TOA CHostPort* pVisibleAddress	The hostport to use in Via and Record-Route headers when the selected sending address is the one identified by opqAddress. Ownership of the pointer is ALWAYS taken from the caller. NULL removes the last set visible address.

Returns

- Success: The visible address has been successfully set.
- Failure: The visible address has NOT been set. This can occur if the opqAddress parameter is invalid.

Description

This method sets and forces the "visible" address for a given local interface to be reported as the content of the pVisibleAddress parameter.

For example, the application has a network interface that uses address 10.1.1.1 and it would like to report it as 10.2.2.2 for some reason. The application will:

1. add the local address 10.1.1.1 through ISipCoreConfig::AddLocalAddress (see page 33) and receive an opaque value to identify it.
2. create a CHostPort (see page 167) instance and set its host part to 10.2.2.2.
3. call ISipCoreConfig::ForceVisibleLocalAddress with the received opaque and the host port instance.

From this point, whenever 10.1.1.1 is used to send requests, the Via will be set to 10.2.2.2.

One scenario where this can be helpful is when a NAT Proxy does not add a Via header for the public address. If the application knows its public address, it can set it properly by using this method.

This method causes the following overrides and behaviours on the SIP-UA config:

- Via and Record-Route headers use the visible address instead of the interface address.
- CSipNetworkInterfaceList::GetLocalAddress returns the forced visible address instead of the interface address/FQDN.
- **The EAddressTypePreference (see page 67) configuration is overridden.**

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.11 - ISipCoreConfig::GetConnectionBlacklistInstance Method New in 4.1.4

Allows access to the persistent connection list module.

C++

```
virtual CSipConnectionBlacklist* GetConnectionBlacklistInstance() = 0;
```

Returns

Pointer to the persistent connection list. NULL if the SIP stack is not initialized yet.

Description

Provides access to the persistent connection list module.

It is through this class that the application can set the manager interface to which the persistent connection list reports its events.

It is also through this class that the application creates and terminates persistent connections.

If a new connection blacklist has been set using SetConnectionBlacklistInstance (see page 44), it is returned here, otherwise the default connection blacklist is returned.

3.2.1.4.2.12 - ISipCoreConfig::GetNetworkInterfaceList Method

Gets the network interfaces address list.

C++

```
virtual mxt_result GetNetworkInterfaceList(OUT CVector<SNetworkIf>& rvecstNetworkIf) = 0;
```

Parameters

Parameters	Description
OUT CVector<SNetworkIf>& rvecstNetworkIf	Local network interfaces addresses list.

Returns

- resS_OK on success.
- resFE_INVALID_STATE: The SIP stack is not properly started.

Description

Returns a list of local network interface addresses. This list contains the local addresses added using the AddLocalAddress (see page 33)() method. For each added local address, it also returns the listening ports added with the ListenA (see page 41) method. If many UDP, TCP, or TLS ports are listening at the same address, it returns only the first registered UDP or TCP port and the first TLS port.

See Also

AddLocalAddress (see page 33), ListenA (see page 41)

3.2.1.4.2.13 - ISipCoreConfig::GetPersistentConnectionList Method

Allows access to the persistent connection list module.

C++

```
virtual CSipPersistentConnectionList* GetPersistentConnectionList() = 0;
```

Returns

Pointer to the persistent connection list. NULL if the SIP stack is not initialized yet.

Description

Provides access to the persistent connection list module.

It is through this class that the application can set the manager interface to which the persistent connection list reports its events.

It is also through this class that the application creates and terminates persistent connections.

3.2.1.4.2.14 - ISipCoreConfig::GetSupportedExtensions Method

Gets the globally supported extension list.

C++

```
virtual void GetSupportedExtensions(OUT const CList<CString>*& rplststrExtension) const = 0;
```

Parameters

Parameters	Description
OUT const CList<CString>*& rplststrExtension	The globally supported extension list.

Description

Gets the globally supported extension list.

The returned list can be NULL, in which case there are no globally supported extension configured.

3.2.1.4.2.15 - ISipCoreConfig::GetSupportedIpVersion Method

This gets the currently configured supported IP version.

C++

```
virtual EIpVersionConfig GetSupportedIpVersion() = 0;
```

Returns

The version(s) that is (are) currently supported.

Description

This gets the currently configured supported IP version.

See Also

ISipCoreConfig::EIpVersionConfig (see page 68)

3.2.1.4.2.16 - ISipCoreConfig::GetTlsContextFactory Method New in 4.1.4

Get the TLS context factory used to create TLS contexts.

C++

```
virtual mxt_result GetTlsContextFactory(OUT ISipTlsContextFactory*& rpTlsContextFactory) = 0;
```

Parameters

Parameters	Description
OUT ISipTlsContextFactory*& rpTlsContextFactory	Returns a pointer to the TLS context factory. A reference is counted for this object when mxt_result is a success.

Returns

- resS_OK: The TLS context factory is returned in the OUT parameter.
- resFE_INVALID_STATE: the transport thread is not initialized.

Description

Returns the TLS context factory used to create TLS contexts. When called for the first time, this method creates the factory if needed. Since the factory is synchronized in the transport thread, the transport thread must be set before calling this method.

The application should configure the TLS context factory at stack startup. However, the TLS context factory can be modified while the stack is running.

See Also

ISipTlsContextFactory (see page 460), SetTransportThread (see page 63)

3.2.1.4.2.17 - ISipCoreConfig::GetUaHelpers Method New in 4.1.7

Allows access to the user agent helpers methods.

C++

```
virtual CSipUaHelpers* GetUaHelpers() const = 0;
```

Returns

Pointer to the user agent helpers object instance.

Description

Provides access to the user agent helpers methods.

3.2.1.4.2.18 - ISipCoreConfig::GetVersion Method

Obtains a NULL-terminated string representing the version of the SIP stack.

C++

```
virtual const char* GetVersion() = 0;
```

Returns

A NULL-terminated string representing the version of the SIP stack.

Description

Gets the version of the SIP stack. The version is a NULL-terminated string that has the following form "major.minor.release.build". CVersion can be used to interpret this string as the major, minor, release, and build numbers.

This can be called at any time.

See Also

CVersion

3.2.1.4.2.19 - ISipCoreConfig::ListenA Method

Listens for incoming SIP packets on an IP address.

C++

```
virtual mxt_result ListenA(IN mxt_opaque opqAddress, IN uint16_t uPort, IN ESipTransport eTransport, IN ISipCoreUser* pCoreUser, IN mxt_opaque opqCallBackParam, OUT mxt_opaque& ropqListen) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqAddress	The address on which to listen, received from AddLocalAddress (see page 33).
IN uint16_t uPort	The port on which to listen.
IN ESipTransport eTransport	The transport to which listen.
IN ISipCoreUser* pCoreUser	Pointer to the interface that is notified once the operation has completed. It can be NULL, in which case the SIP stack does not report any status about the completion of the operation.
IN mxt_opaque opqCallBackParam	Parameters to report when reporting the completion of the operation. This parameter is not used by the SIP stack but simply reported to the pCoreUser upon completion.
OUT mxt_opaque& ropqListen	SIP Stack-specific opaque listen identifier received from ListenA.

Returns

resS_OK: Starts listening, the result is given through EvCommandResult.

resFE_INVALID_ARGUMENT: The provided opqAddress or uPort is invalid.

resFE_INVALID_STATE: The SIP stack is not stated.

Description

This method allows the SIP stack user to have the stack listen on a specific address and port for incoming SIP packets over the specified transport.

Since this is an asynchronous operation performed by the SIP stack, it reports its success or failure through the ISipCoreUser::EvCommandResult (see page 69) method implemented by the SIP stack user. If a failure result code is returned from the method call, no EvCommandResult will be reported.

See Also

StopListeningA (see page 66), AddLocalAddress (see page 33), RemoveLocalAddress (see page 41)

3.2.1.4.2.20 - ISipCoreConfig::RemoveLocalAddress Method

Removes a local address.

C++

```
virtual mxt_result RemoveLocalAddress(IN mxt_opaque opqAddress) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqAddress	SIP Stack-specific opaque address identifier received from AddLocalAddress (see page 33).

Returns

resFE_FAIL: The provided IP address was not present.

resS_OK: The IP address has been removed successfully.

Description

Removes a previously added local address. Calling this also automatically stops all listening on this local address.

It is important to never call this method after Shutdown is called, as Shutdown automatically removes all local addresses and render all associated opqAddress invalid.

See Also

AddLocalAddress (see page 33), StopListeningA (see page 66)

3.2.1.4.2.21 - ISipCoreConfig::RemoveSupportedExtension Method

Removes a globally supported extension.

C++

```
virtual mxt_result RemoveSupportedExtension(IN const CString& rstrExtension) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrExtension	The custom extension to remove.

Returns

resS_OK: The extension was removed. resFE_FAIL: The extension was not found.

Description

Removes a globally supported extension from the list.

3.2.1.4.2.22 - ISipCoreConfig::RemoveTransportObserverA Method

Removes a transport observer from the observer list

C++

```
virtual mxt_result RemoveTransportObserverA(IN ISipTransportObserver* pObserver) = 0;
```

Parameters

Parameters	Description
IN ISipTransportObserver* pObserver	Pointer to the transport observer to remove from the list.

Returns

resS_OK: The transport observer is correctly removed.

resFE_INVALID_STATE: The SIP stack is not properly started.

Description

The method is called to remove a transport observer of activities on sockets from the observer list.

See Also

[ISipTransportObserver](#) (see page 467), [AddTransportObserverA](#) (see page 35)

3.2.1.4.2.23 - ISipCoreConfig::SetApplicationId Method

Sets the application identification to output in User-Agent and Server headers.

C++

```
virtual void SetApplicationId(IN const CString& rstrUserAgentId, IN const CString* pstrServerId = NULL) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrUserAgentId	The application identification to put in the User-Agent header.
IN const CString* pstrServerId = NULL	The application identification to put in the Server header. If NULL, rstrUserAgentId is used.

Description

Sets the application identification to use in the User-Agent and Server headers. This should follow the ABNF defined in RFC 3261 for the Server and User-Agent values.

value	= "Server" HCOLON server-val *(LWS server-val)
server-val	= product / comment
product	= token [SLASH product-version]
product-version	= token
comment	= LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext	= %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII / LWS

By default, this string is empty.

This method can be called at any time.

See Also

[AddStackVersionTold](#) (see page 35), [AddUserAgentToRequests](#) (see page 36), [AddServerToResponses](#) (see page 34)

3.2.1.4.2.24 - ISipCoreConfig::SetClientAuthUnknownQopBehavior Method

Sets the behavior of the ISipDigestClientAuthSvc (see page 474) upon reception of an unknown qop parameter value in the WWW-Authenticate or Proxy-Authenticate header.

C++

```
virtual void SetClientAuthUnknownQopBehavior(IN ISipDigestClientAuthSvc::EUnknownQopBehavior eBehavior) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthSvc::EUnknownQopBehavior eBehavior	The behavior of ISipDigestClientAuthSvc (see page 474) to set.

Description

Sets the behavior of the ISipDigestClientAuthSvc (see page 474) upon reception of an unknown qop parameter value in the WWW-Authenticate or Proxy-Authenticate header. A qop value is considered unknown when it is unsupported by the ISipDigestClientAuthSvc (see page 474) implementation.

See Also

[ISipDigestClientAuthSvc::EUnknownQopBehavior](#) (see page 478)

3.2.1.4.2.25 - ISipCoreConfig::SetCommaSeparatedHeader Method

Sets comma generation setting for the specified header type.

C++

```
virtual mxt_result SetCommaSeparatedHeader(IN ESipHeaderType eHeader, IN ECommaSeparatedHeaderConfig eConfig) = 0;
```

Parameters

Parameters	Description
IN ESipHeaderType eHeader IN ECommaSeparatedHeaderConfig eConfig	Type of header for which to change the comma separated generation behaviour. • eCOMMA_SEPARATED: Headers of the specified type are output as comma-separated headers. • eCRLF_SEPARATED: Headers of the specified type are output as CRLF-separated headers.

Returns

resS_OK: Setting has been applied.

resFE_INVALID_ARGUMENT: This type of header cannot be combined, as per the header definition table.

Description

This method allows the configuration of the behaviour for combining headers when serializing.

The default behaviour is that all headers that can be combined are combined on a single line, with each header separated by a comma and a space.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.26 - ISipCoreConfig::SetConnectionBlacklistInstance Method New in 4.1.4

Allows setting a new persistent connection list module to use, otherwise the default one is used.

C++

```
virtual void SetConnectionBlacklistInstance(TO CSipConnectionBlacklist* pBlacklist) = 0;
```

Parameters

Parameters	Description
TO CSipConnectionBlacklist* pBlacklist	Pointer to the persistent connection list. It can be NULL to release a custom blacklist instance previously set, only if Startup (see page 66) or Shutdown has never been called. Ownership is taken.

Description

Enables the user to set a new connection blacklist instance.

This method is totally optional and does not affect the usage of the stack. If no blacklist instance has been set prior to the stack Startup (see page 66), the stack creates its own default one and manages it.

3.2.1.4.2.27 - ISipCoreConfig::SetConnectionParameters Method

Configures SIP stack connection parameters.

C++

```
virtual mxt_result SetConnectionParameters(IN unsigned int uNumConnectionTarget, IN unsigned int uMaxNumberOfConnections, IN unsigned int uInactivityTimeoutSec) = 0;
```

Parameters

Parameters	Description
IN unsigned int uNumConnectionTarget	The target number of connections that are to be left open by the SIP stack. This must be smaller than or equal to uMaxNumberOfConnections.
IN unsigned int uMaxNumberOfConnections	The maximum number of connections that the SIP stack is allowed to simultaneously manage. In the event that a new connection is required and the SIP stack already manages this number of connections, the least recently used connection is replaced with the new connection.
IN unsigned int uInactivityTimeoutSec	The inactivity timeout, in seconds, to apply to the connections that are over the target number of connections. It should be at least 32 seconds.

Returns

- resS_OK: Parameters properly configured.
- resFE_FAIL: Unable to set configuration parameters.

Description

This method is used to configure how the SIP stack keeps and manages its connections with other SIP devices on the network.

A connection for the SIP stack is a connected socket to a peer. The SIP stack can connect all types of sockets it creates: UDP, TCP, or TLS. The SIP stack obviously uses connected sockets when sending SIP packets over TCP or TLS. The SIP stack also "connects" UDP sockets to a peer when sending a packet over UDP, which allows the detection of some ICMP errors, such as host/port unreachable. The detection of these errors allows faster failover.

A connection to a peer is created when the SIP stack has to send a SIP packet to that peer. Once such a connection is created, the SIP stack manages it according to the description below.

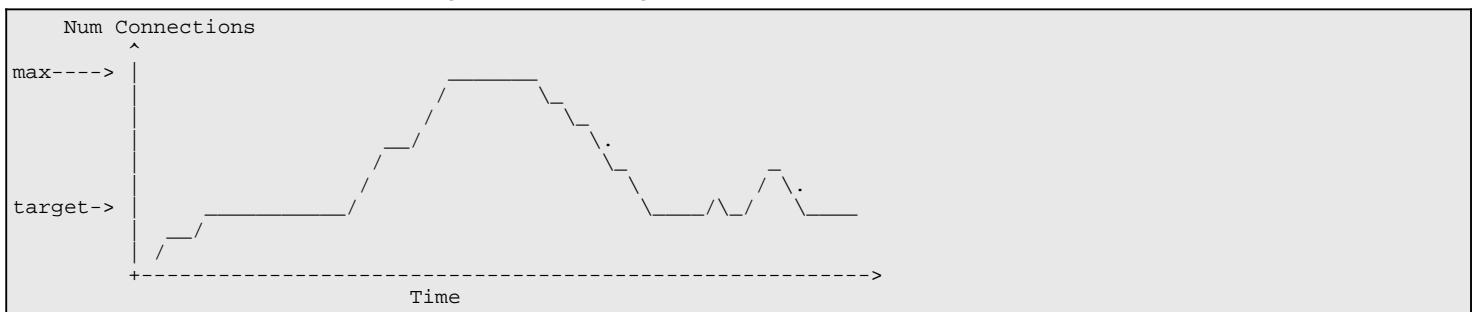
Internally, the SIP stack keeps an ordered list of connections with the most recently used connections appearing first in the list and the least recently used appearing last. Each time a connection is used, its inactivity counter is reset to zero and the connection is placed at the head of the list.

The "uNumConnectionTarget" first connections are never terminated automatically by the SIP stack, regardless of the value of their inactivity counter.

Each connection over uNumConnectionTarget is watched for inactivity timeout. When its inactivity counter reaches uInactivityTimeoutSec, the connection is automatically terminated by the SIP stack.

This has the effect that the SIP stack tends to keep uNumConnectionTarget connections open when in normal runstate. When encountering a busy period, the SIP stack can manage up to uMaxNumberOfConnections, but after this period, the connections over uNumConnectionTarget are released along with their corresponding resources.

The following ASCII graphic shows the SIP stack starting at zero connections, and then building its connection list up to the target number of connections. A busy peak is then encountered, where additional connections are required. Eventually, when the traffic slows down, the number of connections is brought back to the target number of connections.



By default, the SIP stack is initialized with the following values:

- Target number of connections: 5
- Maximum number of connections: 10
- Connection inactivity timeout: 64 seconds

This method can be called at any time after the threading configuration of the SIP stack.

In order to use an non-connected socket for sending SIP packets over UDP, the application must use the `ISipSymmetricUdpSvc` (see page 124). This service allows using the unconnected listening UDP socket for sending SIP packets.

To create a connection to a peer that must always remain established, whether or not there is traffic on it, use the `CSipPersistentConnectionList` and the `ISipPersistentConnectionSvc` (see page 107). These services give control to the SIP stack user on the creation and termination of connections to a peer.

3.2.1.4.2.28 - `ISipCoreConfig::SetCoreThread` Method

Sets the thread used by the SIP core and transactions modules.

C++

```
virtual mxt_result SetCoreThread(IN IEComUnknown* pThread) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pThread	An ECOM object supporting the proper activation, timer, and messaging interfaces. This is usually a CServicingThread object.

Returns

- `resS_OK`: Thread configured successfully.
- `resFE_INVALID_ARGUMENT`: NULL pointer or ECOM instance that does not support the proper interfaces.
- `resFE_UNEXPECTED`: Thread already configured.

Description

This method allows to configure the thread used by the core and transaction modules of the SIP stack.

This thread **MUST** be the one that the application uses when accessing the various User-Agent and Proxy services, and other objects from the core (namely the `ISipContext` (see page 23)).

If the application does not use the `CServicingThread` mechanism for its threading, it must then do the following:

1. Create a `CServicingThread` instance (see examples in `CServicingThread`).
2. Query the `IActivationService` interface.
3. Periodically call the overloaded `Activate` (IN `uint64_t`, OUT `bool*`) function on this interface.
4. Query the `IEComUnknown` interface on the `CServicingThread` instance.
5. Set this `IEComUnknown` interface as the thread for the core and transactions by calling `SetCoreThread`.

This has the effect of periodically providing CPU time from the application's thread to the servicing thread instance and thus to the core and transaction modules.

Depending on the application requirements, a single thread can be used to run more than one of the SIP stack modules (Core, transport, and DNS).

Warning

This method must be called successfully exactly once before starting up the SIP stack. Once set, the thread cannot be changed without a shutdown of the SIP stack.

See Also

[SetTransportThread](#) (see page 63), [SetDnsResolverThread](#) (see page 48), [CServicingThread](#)

3.2.1.4.2.29 - **ISipCoreConfig::SetCoreUser** Method

Configures the core user interface, which receives new packets and chooses how they are handled.

C++

```
virtual mxt_result SetCoreUser(IN ISipCoreUser* pUser) = 0;
```

Parameters

Parameters	Description
IN ISipCoreUser* pUser	Pointer to the core user interface. It can be NULL, in which case no packets are reported. It must not be NULL in normal SIP stack use.

Returns

- **resS_OK**: the value has been correctly updated.
- **resFE_FAIL**: the value could not be updated.

Description

Sets the core user interface. The application using the SIP stack usually implements this interface.

This method can be called at any time after the threading configuration of the SIP stack.

See Also

[ISipCoreUser](#) (see page 69)

3.2.1.4.2.30 - **ISipCoreConfig::SetCSeq64BitsSupport** Method

Sets if a 64 bits sequence number is supported.

C++

```
virtual void SetCSeq64BitsSupport(IN bool bSupported) = 0;
```

Parameters

Parameters	Description
IN bool bSupported	True when a 64bits sequence number must be supported. False for a 32 bits.

Description

Sets if a 64 bits sequence number is supported.

Notes

This method can be called only after the SIP transport thread is set.

3.2.1.4.2.31 - **ISipCoreConfig::SetDefaultDialogMatcherList** Method

Sets the default dialog matcher list.

C++

```
virtual void SetDefaultDialogMatcherList(IN CSipDialogMatcherList& rDialogMatcherList) = 0;
```

Description

Sets the default dialog matcher list that is used when creating the user agent or the session stateful proxy service.

The dialog matcher list is used to keep a list of active dialogs for the above services.

Notes

The application is responsible to create and release the CSipDialogMatcherList. The provided reference to the CSipDialogMatcherList must stay valid until EvShutdownCompleted is fired through the ISipCoreUser (see page 69) interface and after ShutdownA (see page 65) has been called by the application.

See Also

CSipDialogMatcherList, ISipUserAgentSvc (see page 639), ISipSessionStatefulProxySvc (see page 414)

3.2.1.4.2.32 - ISipCoreConfig::SetDnsResolverThread Method

Sets the thread that is used to serially process DNS requests.

C++

```
virtual mxt_result SetDnsResolverThread(IN IEComUnknown* pThread) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pThread	An ECOM object supporting the proper activation, timer, and messaging interfaces. This is usually a CServicingThread object.

Returns

- resS_OK: Thread configured successfully.
- resFE_INVALID_ARGUMENT: NULL pointer or ECOM instance that does not support the proper interfaces.
- resFE_UNEXPECTED: Thread already configured.

Description

This method allows to configure the thread used by the DNS resolving facilities of the SIP stack.

Due to the possible long access time to the network when resolving DNS addresses, this process should be run in its own separate thread, as it blocks the thread it uses while waiting for the DNS answer.

Warning

This method must be called successfully exactly once before starting up the SIP stack. Once set, the thread cannot be changed without a shutdown of the SIP stack.

See Also

SetCoreThread (see page 46), SetTransportThread (see page 63), CServicingThread

3.2.1.4.2.33 - ISipCoreConfig::SetEntityId Method

```
virtual mxt_result SetEntityId(IN mxt_opaque opqListen, IN unsigned int uEntityId) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqListen	SIP stack-specific opaque listen identifier received from ListenA (see page 41).

IN unsigned int uEntityId	The entity ID.
---------------------------	----------------

Returns

- resS_OK: success.
- Other return codes: Failure.

Description

Associate an entity ID with the specified listening opaque. Only SIP Context with the same entity ID can send SIP packets through this interface. This is useful to limit a SIP context to a specific interface. If this method is not called for a listening opaque, 0 is the assumed entity ID.

See Also

ISipContext::SetEntityId (see page 27)

3.2.1.4.2.34 - ISipCoreConfig::SetEnumE164ZoneSuffix Method

Sets the E164 zone suffix for ENUM requests.

C++

```
virtual void SetEnumE164ZoneSuffix(IN const CString& rstrE164ZoneSuffix) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrE164ZoneSuffix	The zone suffix to append to the FQDN form of the AUS. If the string is empty, this parameter defaults to the zone defined by MXD_SIPSTACK_ENUM_DEFAULT_ZONE.

Description

Configures the ENUM zone suffix.

This method can be called at any time after the threading configuration of the SIP stack.

See Also

SipStackCfg.h

3.2.1.4.2.35 - ISipCoreConfig::SetFailoverMode Method

Sets the failover mode.

C++

```
virtual void SetFailoverMode(IN EFailoverMode eFailoverMode) = 0;
```

Parameters

Parameters	Description
IN EFailoverMode eFailoverMode	The failover mode.

Description

Sets the failover mode to use.

The failover mode controls the actions taken when a packet failed to be sent.

The default mode is eFAILOVER_DEFAULT.

Notes

This method can be called only after the DNS resolver thread is set.

Example

```
ISipCoreConfig* pConfig = NULL;
CreateEComInstance(CLSID_CSipCoreConfig, NULL, OUT &pConfig);
```

```

if (pConfig != NULL)
{
    pConfig->SetFailoverMode( ISipCoreConfig::eFAILOVER_DEFAULT ) ;

    pConfig->ReleaseIfRef();
    pConfig = NULL;
}

```

3.2.1.4.2.36 - ISipCoreConfig::SetHandshakeValidatorCallback Method

Configures a callback for the handshake validation.

C++

```
virtual mxt_result SetHandshakeValidatorCallback(IN PFNHandshakeValidator pfnHandshakeValidator, IN bool bOverrideDefaultBehaviour = true) = 0;
```

Parameters

Parameters	Description
IN PFNHandshakeValidator pfnHandshakeValidator	Method used to approve or reject the handshake. PFNHandshakeValidator is a typedef mxt_result (bool blsClient, bool bPeerAuth, const CCertificateChain& rChain, CString& rstrPeerHostname).
IN bool bOverrideDefaultBehaviour = true	True when pfnHandshakeValidator will override the default behaviour of the handshake validator. When false, pfnHandshakeValidator will be invoked prior to the default handshake validator.

Returns

- resS_OK: Parameters properly configured.
- resFE_FAIL: Unable to set configuration parameters.

Description

Configures a callback function that allows the user of the SIP stack to approve or reject the TLS handshake procedure. The following is the SIP stack's default behaviour upon handshake validation:

Handshake approval for TLS server connections:

- Connection is accepted if peer authentication is disabled.
- Connection is dropped if remote peer hostname is not found in the personal certificate.
- Connection is accepted if remote peer hostname is found in the personal certificate.

Handshake approval for TLS client connections:

- Connection is dropped if peer authentication is disabled.
- Connection is dropped if remote peer hostname is not found in the personal certificate.
- Connection is accepted if remote peer hostname is found in the personal certificate.

The parameters of the callback methods are the following:

bIsClient: True if the handshake validation is on a TLS client connection. Otherwise false.

bPeerAuth: True if the remote peer authentication is enabled. Otherwise false.

rChain: The certificate chain returned by the remote peer. This is where the personal certificate is stored.

rstrPeerHostname: The remote peer hostname.

The application can define such a callback to:

- Implement a different logic than the default behaviour described above by setting bOverrideDefaultBehaviour to true.
- Implement a custom pre-processing logic on top of the default behaviour described above by setting bOverrideDefaultBehaviour to false.

The SIP stack calls the configured callback within the context of the SIP transport thread.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.37 - ISipCoreConfig::SetHeaderFormPreference Method

Configures how SIP headers are generated.

C++

```
virtual mxt_result SetHeaderFormPreference(IN EHeaderFormPreference ePreference) = 0;
```

Parameters

Parameters	Description
IN EHeaderFormPreference ePreference	<ul style="list-style-type: none"> • ePREFER_SHORT to use short header form for headers that support it. • ePREFER_LONG to use long header form for all headers.

Returns

resS_OK on success, resFE_FAIL on failure.

Description

Configures how the headers of the packets generated by the SIP stack are generated.

When ePREFER_SHORT is used, headers that support it use the short form. For instance:

- "From:" is output as "f:".
- "Content-Length:" is output as "l:".
- "Supported:" is output as "k:".
- "Max-Forwards:" is output as "Max-Forwards:", as there is no short form version of this header defined.
- etc.

When ePREFER_LONG is used, all headers are output using their long name version.

By default, the SIP stack is initialized with ePREFER_LONG.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.38 - **ISipCoreConfig::SetKeepAliveExtensionMgrA** Method New in 4.1.6

Sets a keep alive extension manager.

C++

```
virtual mxt_result SetKeepAliveExtensionMgrA(IN ISipKeepAliveExtensionMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipKeepAliveExtensionMgr* pMgr	The manager.
Return	resFE_INVALID_STATE: the transport manager is not set.
resS_OK	otherwise.

Description

The application can register this manager when it wants to be informed of non SIP data reception. This is useful when an application wants to send keep alives using STUN requests for instance. When the application is responsible to send STUN requests, it is also responsible to receive and interpret the STUN responses. This is currently possible by configuring this manager.

3.2.1.4.2.39 - **ISipCoreConfig::SetMaxForwards** Method

Sets the value to put in the Max-Forwards header of the requests created by this class.

C++

```
virtual void SetMaxForwards(IN unsigned int uMaxForwards) = 0;
```

Parameters

Parameters	Description
IN unsigned int uMaxForwards	The value to put in the Max-Forwards header of the request created by the CSipPacket (see page 447) class.

Description

Sets the value to put in the Max-Forwards header of the request created by the CSipPacket (see page 447) class. By default, this value is 70, the value recommended by RFC 3261.

3.2.1.4.2.40 - **ISipCoreConfig::SetMaxPayloadSize** Method

Sets the maximum payload size that can be accepted by the SIP stack.

C++

```
virtual mxt_result SetMaxPayloadSize(IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSize	The size, in bytes, of the payload.

Returns

- resS_OK: Max payload size properly configured.
- resFE_FAIL: Unable to set max payload size.

Description

When parsing a received SIP packet, the packet is considered erroneous if the payload is larger than uSize. Therefore it is rejected and an error is generated. A 413 "Request Entity Too Large" response is sent to the peer.

By default, the SIP stack is initialized with a maximum receivable payload size of 1024 bytes.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.41 - ISipCoreConfig::SetMaxReceivePacketSize Method

Sets the maximum receivable size of SIP packet that can be accepted by the SIP stack.

C++

```
virtual mxt_result SetMaxReceivePacketSize(IN unsigned uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned uSize	The size, in bytes, of the receiving packet.

Returns

- resS_OK: Max receive packet size properly configured.
- resFE_FAIL: Unable to set max receive packet size.

Description

When parsing a received SIP packet, the packet is considered erroneous if it is larger than uSize (WITHOUT THE PAYLOAD). Therefore it is rejected and the connection is closed.

Packets received over UDP are not affected, since the maximum UDP packet size is configured at the IP stack level.

By default, the SIP stack is initialized with a maximum receivable packet size of 64K bytes (64 * 1024).

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.42 - ISipCoreConfig::SetMaxSendBufferSize Method

Sets the socket's transmission buffer size.

C++

```
virtual mxt_result SetMaxSendBufferSize(IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSize	The size, in bytes, of the socket's transmission buffer.

Returns

- resS_OK: If properly configured.
- resFE_FAIL: Unable to update buffer size.

Description

This method sets the connections' transmission buffer size. This buffer is used when subsequent SendA are fired on the same connection, which is not yet established or is congested.

There is not a fixed pre-allocated buffer; rather, each time a message must be buffered, a buffer is created and enqueued. Thus, the maximum size corresponds to the maximum accepted for the cumulated sizes of all buffers present in the queue at a given time.

This buffer size is not shared across multiple connections. Each connection can use up to uSize bytes as send buffer.

This buffer size should be at least the same size as the principal buffer, or larger. Otherwise, the SIP stack will not be able to buffer large SIP packets being sent.

By default, the SIP stack is initialized with a maximum send buffer size of 65535 bytes.

This method can be called at any time after the threading configuration of the SIP stack.

See Also

[SetPrincipalBufferSize](#) (see page 55)

3.2.1.4.2.43 - ISipCoreConfig::SetPacketInspectorCallback Method

Configures a callback that the SIP stack calls upon the reception of a complete SIP packet.

C++

```
virtual mxt_result SetPacketInspectorCallback(IN PFNTransportPacketInspector pfnTransportPacketInspector) = 0;
```

Parameters

Parameters	Description
IN PFNTransportPacketInspector pfnTransportPacketInspector	Method used to modify the packet. PFNTransportPacketInspector is a typedef void (CSipPacketParser::SRawData (see page 313)*&)

Returns

- resS_OK: Parameters properly configured.
- resFE_FAIL: Unable to set configuration parameters.

Description

Configures a callback that the SIP stack calls upon the reception of a complete SIP packet.

The SIP stack provides the received data as raw data. The callback can modify the received data before the data is actually parsed, when the callback returns.

The SIP stack calls the configured callback within the context of the SIP transport thread.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.44 - ISipCoreConfig::SetPacketModifierCallback Method

Configures a callback that the SIP stack calls upon the reception and parsing of a complete SIP packet.

C++

```
virtual mxt_result SetPacketModifierCallback(IN PFNTransportPacketModifier pfnTransportPacketModifier) = 0;
```

Parameters

Parameters	Description
IN PFNTransportPacketModifier pfnTransportPacketModifier	Method used to modify the packet. PFNTransportPacketModifier is a typedef void (CSipPacket (see page 447)&)

Returns

- resS_OK: Parameters properly configured.
- resFE_FAIL: Unable to set configuration parameters.

Description

Configures a callback that the SIP stack calls upon the reception of a complete SIP packet.

The SIP stack provides a parsed SIP packet. The callback should modify the headers of the provided SIP packet before the packet is handled by the SIP stack.

The callback MUST NOT modify the following mandatory header:

- From
- To
- Call-ID

- Via
- CSeq

This callback is not called for packets that were received and discarded because of invalid above mandatory header.

The SIP stack calls the configured callback within the context of the SIP transport thread. Any packet modifications MUST be done synchronously.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.45 - **ISipCoreConfig::SetPrincipalBufferSize** Method

Sets the size of the principal buffer.

C++

```
virtual mxt_result SetPrincipalBufferSize(IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSize	The size, in bytes, of the memory to allocate for the principal buffer.

Returns

- resS_OK: Updated successfully.
- resFE_FAIL: Unable to update buffer size.

Description

This method is used to reserve a memory space for the allocation of the principal buffer. This buffer is the region from where raw SIP packets are sent and where they are received. Only one such buffer is used for the whole SIP stack.

The buffer size should correspond to the maximum packet size (including the payload) the application is ready to receive or send over UDP. The following are specifications from RFC 3261:

However, implementations MUST be able to handle messages up to the maximum datagram packet size. For UDP, **this** size is 65 535 bytes including IP and UDP headers.

By default, the SIP stack is initialized with a principal buffer size of 65535 bytes.

This method can be called at any time after the threading configuration of the SIP stack.

3.2.1.4.2.46 - **ISipCoreConfig::SetSipDataLogger** Method

Configures the data logger.

C++

```
virtual void SetSipDataLogger(IN ISipDataLogger* pDataLogger) = 0;
```

Parameters

Parameters	Description
IN ISipDataLogger* pDataLogger	The reference to the data logger.

Description

Configures the data logger that is used for logging SIP packets and raw data sent and received from the network.

The SIP stack does NOT take ownership of the given data logger. Thus, the application must make sure the pointer stays valid as long as the stack knows it.

The application is also responsible for freeing the resources of the given data logger.

Notes

This method can be called only after the SIP transport thread is set.

3.2.1.4.2.47 - ISipCoreConfig::SetSocketClosureType Method New in 4.1.5

```
virtual mxt_result SetSocketClosureType(IN ISocket::ECloseBehavior eCloseBehavior) = 0;
```

Parameters

Parameters	Description
IN ISocket::ECloseBehavior eCloseBehavior	The closing behaviour to use.

Returns

resS_OK: Successfully set the closing behaviour.

resFE_INVALID_STATE: The stack has not been configured properly yet.

Description

This sets the closing behaviour to use to close connected client sockets while removing a network interface. This closing behaviour is also used if ShutdownA (see page 65) is called and some network interfaces are still present. By default, graceful closes are done.

3.2.1.4.2.48 - ISipCoreConfig::SetSpirallingSvcState Method

Sets the Spiralling Service state.

C++

```
virtual mxt_result SetSpirallingSvcState(IN ESpirallingSvcState eState) = 0;
```

Parameters

Parameters	Description
IN ESpirallingSvcState eState	The new state of the Spiralling Service.

Returns

resS_OK: The new Spiralling Service state is configured.

resFE_INVALID_STATE: The SIP stack has been started.

Description

Configures whether or not the Spiralling Service is enabled.

This method can be called only before the startup of the SIP stack.

3.2.1.4.2.49 - ISipCoreConfig::SetSupportedDnsQueries Method

Sets the type(s) of DNS queries supported by the application.

C++

```
virtual void SetSupportedDnsQueries(IN int nDnsQueryBitset) = 0;
```

Parameters

Parameters	Description
IN int nDnsQueryBitset	The DNS query type(s) the SIP stack may use. This value can be a concatenation of various values created by using the bitwise "OR" operator on the ESipDnsQuery enumeration.

Description

This method sets the type(s) of DNS queries that the SIP stack supports and uses. It is possible to disable NAPTR AND SRV queries or only NAPTR queries. The sample code below shows how to perform those operations.

By default, NAPTR and SRV records are active.

This method can be called at any time after the DNS resolver thread is set.

NOTES: Since NAPTR queries require SRV queries, it is not possible to enable NAPTR without also enabling SRV. The SIP stack automatically handles this case.

In all cases, the SIP stack always supports A queries and uses them as needed.

See Also

ESipDnsQuery in CServerLocator.h

Example

```
ISipCoreConfig* pConfig = NULL;

CreateEComInstance(CLSID_CSipCoreConfig, NULL, OUT &pConfig);

if (pConfig != NULL)
{
    // To disable NAPTR and SRV queries at once, call:
    //-----
    pConfig->SetSupportedDnsQueries(eNONE);

    // To enable only NAPTR queries, call:
    // (As a side effect, SRV queries are also enabled).
    //-----
    pConfig->SetSupportedDnsQueries(eNAPTR);

    // To enable only SRV queries, call:
    //-----
    pConfig->SetSupportedDnsQueries(eSRV);

    // To explicitly enable both types of queries, call:
    //-----
    pConfig->SetSupportedDnsQueries(eNAPTR | eSRV);
    pConfig->SetSupportedDnsQueries(static_cast<ESipDnsQuery>(eNAPTR | eSRV));

    // Release the ISipCoreConfig reference.
    //-----
    pConfig->ReleaseIfRef();
}
```

3.2.1.4.2.50 - ISipCoreConfig::SetSupportedIpVersion Method

Configures which versions of TCP/IP are supported or enabled by the application.

C++

```
virtual void SetSupportedIpVersion(IN EIPEndPointConfig eIpVersion) = 0;
```

Parameters

Parameters	Description
IN EIPEndPointConfig eIpVersion	The version(s) that the application wants to support.

Description

This configures whether the application wants to support only IPv4, only IPv6, or both IPv4 and IPv6. This is then used by the SIP stack to determine if it needs to perform A or AAAA queries when resolving addresses. Note that MXD_SIPSTACK_IPV6_ENABLE_SUPPORT (see page 14) must be defined to be able to use IPv6. If MXD_SIPSTACK_IPV6_ENABLE_SUPPORT (see page 14) is not defined, IPv4 is enabled by default. If it is defined, IPv4 and IPv6 are enabled by default.

Notes

This method can be called only after the DNS resolver thread is set.

See Also

ISipCoreConfig::EIPEndPointConfig (see page 68), MXD_SIPSTACK_IPV6_ENABLE_SUPPORT (see page 14)

Example

```
ISipCoreConfig* pConfig = NULL;

CreateEComInstance(CLSID_CSipCoreConfig, NULL, OUT &pConfig);

if (pConfig != NULL)
{
    pConfig->SetSupportedIpVersion(ISipCoreConfig::eCONFIG_IPV4);
```

```

    pConfig->ReleaseIfRef();
}

```

3.2.1.4.2.51 - ISipCoreConfig::SetSupportedSipTransport Method

Sets the transport protocol the SIP stack may use for sending SIP packets.

C++

```
virtual void SetSupportedSipTransport(IN int nTransportBitset) = 0;
```

Parameters

Parameters	Description
IN int nTransportBitset	The transport(s) the SIP stack may use. This value can be a concatenation of various values created by using the bitwise "OR" operator on the ESipTransport enumeration.

Description

Sets the SIP transport protocol the SIP stack may use for sending SIP packets. Multiple transports can be concatenated to set them all at once. The previously configured value is lost.

By default, eUDP, eTCP, and eTLS are supported.

This method can be called at any time after the DNS resolver thread is set.

See Also

ESipTransport in CSipTransportTools.

Example

```

ISipCoreConfig* pConfig = NULL;

CreateEComInstance(CLSID_CSipCoreConfig, NULL, OUT &pConfig);

if (pConfig != NULL)
{
    pConfig->SetSupportedSipTransport(static_cast<ESipTransport>(eTCP | eUDP));
    pConfig->ReleaseIfRef();
}

```

3.2.1.4.2.52 - ISipCoreConfig::SetT1 Method

Overrides the value to use for T1.

C++

```
virtual mxt_result SetT1(IN unsigned int uT1Ms) = 0;
```

Parameters

Parameters	Description
IN unsigned int uT1Ms	The value to use for T1 in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for T1. By default, the SIP stack uses a value of 500 ms, as specified in RFC 3261.

One should modify this value with care as it affects the retransmission interval of requests and responses on unreliable transports.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

3.2.1.4.2.53 - ISipCoreConfig::SetT2 Method

Overrides the value to use for T2.

C++

```
virtual mxt_result SetT2(IN unsigned int uT2Ms) = 0;
```

Parameters

Parameters	Description
IN unsigned int uT2Ms	The value to use for T2 in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for T2. By default, the SIP stack uses a value of 4000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the retransmission interval of non-INVITE requests and INVITE responses on unreliable transports.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

3.2.1.4.2.54 - ISipCoreConfig::SetT4 Method

Overrides the value to use for T4.

C++

```
virtual mxt_result SetT4(IN unsigned int uT4Ms) = 0;
```

Parameters

Parameters	Description
IN unsigned int uT4Ms	The value to use for T4 in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for T4. By default, the SIP stack uses a value of 5000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the maximum duration a message remains in the network on unreliable transports.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

3.2.1.4.2.55 - ISipCoreConfig::SetTimeoutTimer Method

Sets the timeout value used by some SIP timers and timeouts.

C++

```
virtual mxt_result SetTimeoutTimer(IN unsigned int uTimeoutTimerMs) = 0;
```

Parameters

Parameters	Description
IN unsigned int uTimeoutTimerMs	The value to use for the timeout timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the timeout value used for the SIP timers and timeouts described below. Its default value is 32 seconds as per RFC 3261.

- Timer B: INVITE transaction timeout timer (RFC 3261, section 17.1.1.2).
- Timer F: non-INVITE transaction timeout timer (RFC 3261, section 17.1.2.2).
- Timer H: Wait time for ACK receipt (RFC 3261, section 17.2.1).
- Timer J: Wait time for non-INVITE request retransmits (RFC 3261, section 17.2.2).
- UAS INVITE 2xx retransmission timeout until the ACK is received. (RFC 3261, section 13.3.1.4).
- UAC INVITE sent ACK buffered timeout. When UAS retransmits 2xx, the buffered ACK is sent again (13.2.2.4).
- UAC final response waiting timeout after the first 2xx is received on early dialog.
- PROXY timeout period that final responses to an INVITE request are forwarded. (RFC 3261, section 16.7, bullet 5).
- PROXY retransmission timeout period to received 2xx, ACK and 1xx reliable (ISipSessionStatefulProxySvc (see page 414)).
- Subscribe final NOTIFY waiting timeout.

When timers B, F, H or J are set with a value > 0 using its specific setter (with ISipCoreConfig::SetTimerB (see page 60)(value) for timer B), this new value will be used instead of the SetTimeoutTimer value. The other timeouts described above (other timers than B, F, H and J), always use the SetTimeoutTimer value.

One should modify this value with care as it affects the time needed before a transaction waiting for a response times out.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

3.2.1.4.2.56 - ISipCoreConfig::SetTimerB Method

Overrides the value to use for the Timer B (as specified in RFC 3261).

C++

```
virtual mxt_result SetTimerB(IN unsigned int uTimerMs) = 0;
```

Parameters

Parameters	Description
uTimeoutMs	The value to use for the timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for the timer. By default, the SIP stack uses a value of 32000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the INVITE transaction timeout.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

See Also

[SetT1](#) (see page 58) [SetT2](#) (see page 59) [SetT4](#) (see page 59) [SetTimeoutTimer](#) (see page 59) [SetTimerD](#) (see page 61)
[SetTimerF](#) (see page 61) [SetTimerH](#) (see page 62) [SetTimerJ](#) (see page 62)

3.2.1.4.2.57 - ISipCoreConfig::SetTimerD Method

Overrides the value to use for the Timer D (as specified in RFC 3261).

C++

```
virtual mxt_result SetTimerD(IN unsigned int uTimerMs) = 0;
```

Parameters

Parameters	Description
uTimeoutMs	The value to use for the timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for the timer. By default, the SIP stack uses a value of 32000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the wait time for response retransmits.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

See Also

[SetT1](#) (see page 58) [SetT2](#) (see page 59) [SetT4](#) (see page 59) [SetTimeoutTimer](#) (see page 59) [SetTimerB](#) (see page 60)
[SetTimerF](#) (see page 61) [SetTimerH](#) (see page 62) [SetTimerJ](#) (see page 62)

3.2.1.4.2.58 - ISipCoreConfig::SetTimerF Method

Overrides the value to use for the Timer F (as specified in RFC 3261).

C++

```
virtual mxt_result SetTimerF(IN unsigned int uTimerMs) = 0;
```

Parameters

Parameters	Description
uTimeoutMs	The value to use for the timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for the timer. By default, the SIP stack uses a value of 32000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the non-INVITE transaction timeout timer.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

See Also

SetT1 (see page 58) SetT2 (see page 59) SetT4 (see page 59) SetTimeoutTimer (see page 59) SetTimerB (see page 60) SetTimerD (see page 61) SetTimerH (see page 62) SetTimerJ (see page 62)

3.2.1.4.2.59 - ISipCoreConfig::SetTimerH Method

Overrides the value to use for the Timer H (as specified in RFC 3261).

C++

```
virtual mxt_result SetTimerH(IN unsigned int uTimerMs) = 0;
```

Parameters

Parameters	Description
uTimeoutMs	The value to use for the timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for the timer. By default, the SIP stack uses a value of 32000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the wait time for ACK receipts.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

See Also

SetT1 (see page 58) SetT2 (see page 59) SetT4 (see page 59) SetTimeoutTimer (see page 59) SetTimerB (see page 60) SetTimerD (see page 61) SetTimerF (see page 61) SetTimerJ (see page 62)

3.2.1.4.2.60 - ISipCoreConfig::SetTimerJ Method

Overrides the value to use for the Timer J (as specified in RFC 3261).

C++

```
virtual mxt_result SetTimerJ(IN unsigned int uTimerMs) = 0;
```

Parameters

Parameters	Description
uTimeoutMs	The value to use for the timer in ms.

Returns

- resS_OK: the value has been correctly updated.
- resFE_FAIL: the value could not be updated.

Description

Sets the value to use for the timer. By default, the SIP stack uses a value of 32000 ms (as specified in RFC 3261).

One should modify this value with care as it affects the wait time for non-INVITE request retransmits on unreliable transports.

This method can be called at any time after the threading configuration of the SIP stack. Note that the new value does not affect ongoing transactions.

See Also

SetT1 (see page 58) SetT2 (see page 59) SetT4 (see page 59) SetTimeoutTimer (see page 59) SetTimerB (see page 60)
SetTimerD (see page 61) SetTimerF (see page 61) SetTimerH (see page 62)

3.2.1.4.2.61 - ISipCoreConfig::SetTlsSessionCacheMaxSize Method

Sets the maximum size of the cached TLS session list.

C++

```
virtual mxt_result SetTlsSessionCacheMaxSize(IN uint32_t uTlsSessionCacheMaxSize) = 0;
```

Parameters

Parameters	Description
IN uint32_t uTlsSessionCacheMaxSize	Maximum size for the list of cached TLS session.

Returns

- resS_OK: Cached TLS session list maximum size set.
- resFE_FAIL: Unable to set cached TLS session list maximum size.
- resFE_INVALID_ARGUMENT: Maximum size must be at least 1.

Description

Sets the maximum size of the cached TLS session list. The maximum size must be at least 1.

3.2.1.4.2.62 - ISipCoreConfig::SetTransportThread Method

Sets the thread that is used by the SIP transport module and the sockets it uses.

C++

```
virtual mxt_result SetTransportThread(IN IEComUnknown* pThread) = 0;
```

Parameters

Parameters	Description
IN IEComUnknown* pThread	An ECOM object supporting the proper activation, timer, and messaging interfaces. This is usually a CServicingThread object.

Returns

- **resS_OK:** Thread configured successfully.
- **resFE_INVALID_ARGUMENT:** NULL pointer or ECOM instance that does not support the proper interfaces.
- **resFE_UNEXPECTED:** Thread already configured.

Description

This method allows to configure the thread used by the SIP transport and the socket it uses.

Depending on the application requirements, a single thread can be used to run more than one of the SIP stack modules (Core, transport, and DNS).

Warning

This method must be called successfully exactly once before starting up the SIP stack. Once set, the thread cannot be changed without a shutdown of the SIP stack.

See Also

[SetCoreThread](#) (see page 46), [SetDnsResolverThread](#) (see page 48), [CServicingThread](#)

3.2.1.4.2.63 - ISipCoreConfig::SetUaResponseMultipleViasCheck Method

Sets the multiple Via headers check in incoming responses destined to user-agents.

C++

```
virtual mxt_result SetUaResponseMultipleViasCheck( IN bool bCheckMultipleVias, IN TOA CVector<CString>* pvecstrProxyIds = NULL) = 0;
```

Parameters

Parameters	Description
IN bool bCheckMultipleVias	True if the transport needs to check for multiple Vias in user-agent responses. If false nothing is done, this is the default.
IN TOA CVector<CString>* pvecstrProxyIds = NULL	This needs to be set when the application has at least one user-agent and one proxy within the same stack instance. If the application is only a UA, this must be NULL. This must contain the proxy IDs as set in the CSipProxyConfig::GetId method. If the application has only one proxy ID, then the vector must contain only one ID. When there are many proxies, it is possible that there is one ID per proxy.

Returns

- **resS_OK:** success.
- **resFE_FAIL:** the transport thread is not set or pvecstrProxyIds is empty.

Description

This method is useful only when the application is acting as a user-agent. It is used to drop responses that contain more than one Via header within an incoming response. When enabled (bCheckMultipleVias set to true), all responses destined to a user-agent are dropped (discarded). This is to be compliant to section 8.1.1.3 of the RFC 3261.

If this setting is not enabled, then the responses containing more than one Via header are processed as usual: e.g. transport observers see such responses even if there are more than one Via.

When the application is acting as a proxy, this method must not be called. If the application is acting both as a proxy and a user-agent, the method must be called only if the application wants to discard the responses containing multiple Vias.

When acting as a proxy, the stack can receive responses with more than one Via. This is because the proxy adds a Via header before forwarding a request. The proxy encodes specific information when generating the Via. Hence it is possible to determine if a Via was generated by the stack or not. This means that when bCheckMultipleVias is true and pvecstrProxyIds has at least one proxy ID, the

response is discarded only when the Via was not generated by the stack.

By default, the check for multiple Vias is disabled

See Also

CSipProxyConfig::GetProxyId (see page 404)

3.2.1.4.2.64 - ISipCoreConfig::SetUdpMaxSizeThreshold Method New in 4.1.4

Sets the threshold above which a request is first tried using TCP when no transport mechanism is specified (as per RFC 3261 section 18.1.1).

C++

```
virtual void SetUdpMaxSizeThreshold(IN unsigned uSize) = 0;
```

Parameters

Parameters	Description
IN unsigned uSize	The size, in bytes, of the UDP maximum size threshold. Zero (0) has a special meaning, effectively setting an infinite size and thus disabling the UDP maximum size threshold.

Description

RFC 3261 section 18.1.1 specifies that a request within 200 bytes of the path MTU, or larger than 1300 bytes is the path MTU is unknown, MUST be sent using a congestion controlled transport protocol, such as TCP.

This method sets the threshold above which this TCP override happens. It defaults to 1300 bytes.

As specified in RFC 3261, if the request is sent using TCP because of this size constraint but would otherwise have been sent using UDP, and that request subsequently fails, it is retried using UDP.

The server location service uses the value specified by this method to determine the maximum size a packet should have, given its transport protocol. When the server location service is not enabled, the maximum size defaults to infinite. However, this should not happen under normal circumstances because all use cases require this service to be enabled.

Notes

This method can be called only after the SIP core thread is set.

3.2.1.4.2.65 - ISipCoreConfig::SetViaAddressType Method

Configures which type of addresses the SIP stack uses in Via headers.

C++

```
virtual void SetViaAddressType(IN EAddressTypePreference ePreference) = 0;
```

Parameters

Parameters	Description
IN EAddressTypePreference ePreference	The configuration for the Via address.

Description

Configures how the SIP stack builds the Via addresses from the information configured by AddLocalAddress (see page 33). ePREFER_FQDN is the default value.

See Also

AddLocalAddress (see page 33)

3.2.1.4.2.66 - ISipCoreConfig::ShutdownA Method

Shuts down the SIP stack asynchronously.

C++

```
virtual void ShutdownA() = 0;
```

Description

Shuts down the SIP stack. When shutdown process is completed, `EvShutdownCompleted` is called on the `ISipCoreUser` (see page 69) interface.

See Also

`Startup` (see page 66)

3.2.1.4.2.67 - `ISipCoreConfig::Startup` Method

Starts the initialization procedures of the SIP stack.

C++

```
virtual mxt_result Startup() = 0;
```

Returns

`resS_OK`: If it successfully started.

`resFE_FAIL`: If it is already started or if it failed to be started.

Description

Starts the SIP stack by initializing the necessary components.

See Also

`ShutdownA` (see page 65)

3.2.1.4.2.68 - `ISipCoreConfig::StopListeningA` Method

Stops listening for incoming SIP packets on an IP address.

C++

```
virtual mxt_result StopListeningA(IN mxt_opaque opqListen, IN ISipCoreUser* pCoreUser, IN mxt_opaque opqCallBackParam) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqListen	The listen identifier on which to stop listening, received from <code>ListenA</code> (see page 41).
IN ISipCoreUser* pCoreUser	Pointer to the interface that is notified once the operation has completed. It can be <code>NULL</code> , in which case the SIP stack does not report any status about the completion of the operation.
IN mxt_opaque opqCallBackParam	Parameters to report when reporting the completion of the operation. This parameter is not used by the SIP stack but simply reported to the <code>pCoreUser</code> upon completion.

Returns

`resS_OK`: Stops listening, the result is given through `EvCommandResult`.

`resFE_INVALID_STATE`: The SIP stack is not stated or this listen identifier is already proceeding `StopListeningA`.

`resFE_INVALID_ARGUMENT`: The provided `opqListen` parameter is invalid.

Description

This method allows the SIP stack user to stop a previous call to `ListenA` (see page 41).

Since this is an asynchronous operation performed by the SIP stack, it reports its success or failure through the `ISipCoreUser::EvCommandResult` (see page 69) method implemented by the SIP stack user. If a failure result code is returned from the method call, no `EvCommandResult` will be reported.

See Also

ListenA (see page 41), AddLocalAddress (see page 33), RemoveLocalAddress (see page 41)

3.2.1.4.2.69 - ISipCoreConfig::UpdateParserGrammar Method

Updates the grammar of a given character set.

C++

```
virtual mxt_result UpdateParserGrammar(IN const unsigned int uIndex, IN const bool bValue, IN const
CToken::ECharSet eCharSet) = 0;
```

Parameters

Parameters	Description
IN const unsigned int uIndex	Index of the value to change in eCharSet.
IN const bool bValue	Whether or not the character at the specified index in the specified character set is legal.
IN const CToken::ECharSet eCharSet	Character set in which to change a grammar value.

Returns

- resS_OK upon success.
- resFE_INVALID_ARGUMENT if eCharSet or uIndex is invalid.

Description

Changes whether or not an element is legal in a specified character set. An element's index is the same as in an ASCII table. For more information on the current character set, please refer to RFC 3261 section 25.1.

Warning

Changing the grammar in any way can have important side effects and may potentially cause crashes. Media5 should be consulted before doing so.

Notes:

Must be called before ISipCoreConfig::Startup (see page 66).

3.2.1.4.3 - Enumerations**3.2.1.4.3.1 - ISipCoreConfig::EAddressTypePreference Enumeration**

Enumerations describing the preference for how addresses are represented.

C++

```
enum EAddressTypePreference {
    ePREFER_FQDN,
    eFORCE_IP_ADDRESS
};
```

Members

Members	Description
ePREFER_FQDN	Addresses are represented as an FQDN whenever possible.
eFORCE_IP_ADDRESS	For the address to an IP address, even if there is an FQDN associated with it.

3.2.1.4.3.2 - ISipCoreConfig::ECommaSeparatedHeaderConfig Enumeration

Configuration options for SetCommaSeparatedHeader (see page 43).

C++

```
enum ECommaSeparatedHeaderConfig {
    eCOMMA_SEPARATED,
    eCRLF_SEPARATED
};
```

Members

Members	Description
eCOMMA_SEPARATED	Use comma separated headers for specified header.
eCRLF_SEPARATED	Place each header on separated lines.

3.2.1.4.3.3 - ISipCoreConfig::EFailoverMode Enumeration

```
enum EFailoverMode {
    eFAILOVER_DEFAULT,
    eFAILOVER_NO_TCP_ASSUMPTION
};
```

Description

These are the possible modes for transport failover. The mode is used to decide what needs to be done when a packet fails to be sent.

Members

Members	Description
eFAILOVER_DEFAULT	This is the default value. The default failover is to assume that TCP is supported by the peer.
eFAILOVER_NO_TCP_ASSUMPTION	TCP failover only occurs if the peer supports TCP according to the information provided by the DNS configuration and the DNS answer. If the peer does not support TCP, then no failover will occur from UDP to TCP.

3.2.1.4.3.4 - ISipCoreConfig::EHeaderFormPreference Enumeration

Configuration options for SetHeaderFormPreference (see page 51).

C++

```
enum EHeaderFormPreference {
    ePREFER_SHORT,
    ePREFER_LONG
};
```

Members

Members	Description
ePREFER_SHORT	Prefer using short header form.
ePREFER_LONG	Prefer using long header form.

3.2.1.4.3.5 - ISipCoreConfig::EIpVersionConfig Enumeration

Enumeration defining the possible combination of supported versions of IP.

C++

```
enum EIIPVersionConfig {
    eCONFIG_IPV4,
    eCONFIG_IPV6,
    eCONFIG_IPV4_AND_IPV6
};
```

Members

Members	Description
eCONFIG_IPV4	The application only supports IPv4.
eCONFIG_IPV6	The application only supports IPv6.
eCONFIG_IPV4_AND_IPV6	The application supports both IPv4 and IPv6.

3.2.1.4.3.6 - ISipCoreConfig::ESpirallingSvcState Enumeration

Enumeration defining the Spiralling Service state.

C++

```
enum ESpirallingSvcState {
    eACTIVE,
    eINACTIVE
};
```

Members

Members	Description
eACTIVE	The Spiralling Service is active.
eINACTIVE	The Spiralling Service is inactive.

3.2.1.5 - ISipCoreUser Class

Class Hierarchy

```
ISipCoreUser
```

C++

```
class ISipCoreUser;
```

Description

This interface represents, for the SIP stack, the user of the SipCore module. It is used to report the reception of incoming SIP packets, the shutdown completion, and the asynchronous result of the ListenA and StopListeningA methods.

This interface is implemented by the application using the M5T SIP Stack.

Location

SipCore/ISipCoreUser.h

Methods

Method	Description
◆ A EvCommandResult (see page 69)	Notifies the core user of the result of an operation.
◆ A EvOnPacketReceived (see page 70)	A packet has been received by the stack.
◆ A EvShutdownCompleted (see page 70)	The ISipCoreConfig (see page 28) shutdown process has been completed.

Legend

◆	Method
A	abstract

3.2.1.5.1 - Methods

3.2.1.5.1.1 - ISipCoreUser::EvCommandResult Method

Notifies the core user of the result of an operation.

C++

```
virtual void EvCommandResult(IN mxt_result res, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_result res	The result code. For further details, see the async method reporting this event upon completion or error.
IN mxt_opaque opq	Opaque parameter given to the core config.

Description

This event is reported by the SIP stack to a specific core user when an operation completes, successfully or not.

See Also

ISipCoreConfig::ListenA (see page 41), ISipCoreConfig::StopListeningA (see page 66)

3.2.1.5.1.2 - ISipCoreUser::EvOnPacketReceived Method

A packet has been received by the stack.

C++

```
virtual void EvOnPacketReceived(IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The incoming packet. The event implementation must properly manage the packet's reference count.

Description

This event is reported to the SipCore user when a new incoming packet is received and was not handled by the lower layers of the stack.

Upon receiving a SIP packet, the application can choose to have the packet handled by UA or proxy service, usually based on the request-URI and the existence of a dialog for this packet. Furthermore, it can use the existing authentication service for verifying the credentials associated with the packet.

The following are usually handled by the lower layers of the stack:

- Request retransmissions
- Response retransmissions, except 200 OK to INVITEs
- Incoming CANCEL requests
- Incoming ACK to negative final responses

Upon reception of this event, the SipCore user usually implements the following steps:

- Authentication: The user can choose or not to verify the authentication information received with the packet. This optional step is done with the stateless digest server authentication service (ISipStatelessDigestServerAuthSvc (see page 110)).
- Existing dialog verification: If the authentication information is valid or if the user decided not to verify it, it should then check if the incoming packet matches an existing dialog. This is done with the dialog matcher list service (CSipDialogMatcherList).
- If the packet does not match an existing dialog, the user can choose how to handle the packet. It usually does so based on the request-URI of incoming requests. For users that support both proxy and UA services, it can choose to proxy the request or handle it as a UA. This decision is really application specific.

This event is always reported within the implementor's SipCore thread, thus it usually requires no synchronization and can be handled synchronously.

3.2.1.5.1.3 - ISipCoreUser::EvShutdownCompleted Method

The ISipCoreConfig (see page 28) shutdown process has been completed.

C++

```
virtual void EvShutdownCompleted() = 0;
```

Description

The shutdown process of the ISipCoreConfig (see page 28) object has been completed. This means that the stack has been stopped.

3.2.1.6 - ISipDialogMatcher Class

Class Hierarchy



C++

```
class ISipDialogMatcher : public IEComUnknown;
```

Description

This interface is used by the CSipDialogMatcherList to handle the packet only when its dialog identification matches that of this object.

Location

SipCore/ISipDialogMatcher.h

See Also

CSipDialogMatcherList

Methods

Method	Description
◆ A GetDialogMatcherList (see page 71)	Obtains the dialog matcher list.
◆ A OnPacketReceived (see page 71)	Handles the packet with the parent context if it belongs to this dialog.
◆ A SetDialogMatcherList (see page 72)	Sets the dialog matcher list.

Legend

◆ A	Method
A	abstract

3.2.1.6.1 - Methods

3.2.1.6.1.1 - ISipDialogMatcher::GetDialogMatcherList Method

Obtains the dialog matcher list.

C++

```
virtual CSipDialogMatcherList* GetDialogMatcherList() = 0;
```

Returns

Pointer to the configured CSipDialogMatcherList. It can return NULL when called before SetDialogMatcherList (see page 72) or when ISipCoreConfig::SetDefaultDialogMatcherList (see page 47) has not been called.

Description

Returns a pointer to configured CSipDialogMatcherList.

See Also

SetDialogMatcherList (see page 72), CSipDialogMatcherList

3.2.1.6.1.2 - ISipDialogMatcher::OnPacketReceived Method

Handles the packet with the parent context if it belongs to this dialog.

C++

```
virtual mxt_result OnPacketReceived(IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The received packet to handle.

Returns

- **resFE_INVALID_STATE**: The dialog state does not enable this service to handle this packet. It probably means that the packet belongs to another dialog.
- **resFE_FAIL**: The packet belongs to this dialog but none of the services attached to the same context as this service were able to handle the packet.
- **resS_OK**: The packet was correctly handled. Either the service that handled it automatically processed it or an event was reported through its manager interface to inform the application of the action to take. It can also be that the packet was a retransmission of a packet handled by a transaction attached to the same context, in which case the packet is silently ignored.

Description

This method verifies if the received packet belongs to the dialog this service handles, in which case the packet is handled by its parent SIP Context.

Dialog matching rules are based on RFC 3261, section 12.2.2. The Call-ID header, the To header Tag parameter, and the From header Tag parameter are used. Note that when this service is uninitialized, it considers that it can handle any packet.

After determining that the packet belongs to this dialog, the implementation should verify that the CSeq number is not out of order. If it is, it should create a request context and a CSipUserAgentReqCtxSvc to put as its owner and make that request context handle the packet. The CSipUserAgentReqCtxSvc automatically answers with a "500 CSeq Number Out of Order" response. Note that out of order CSeq should apply only to requests within a dialog.

If the packet's CSeq number is in sequence or the packet belongs to this dialog, it is given to the parent ISipContext::OnPacketReceived (see page 27) method. An owner is determined and the packet is given to it.

3.2.1.6.1.3 - ISipDialogMatcher::SetDialogMatcherList Method

Sets the dialog matcher list.

C++

```
virtual void SetDialogMatcherList(IN CSipDialogMatcherList& rDialogMatcherList) = 0;
```

Description

Sets the dialog matcher list that is used to keep a list of active dialogs for services that implement that interface.

When adding the user agent service or the session stateful proxy service to a context, the application is responsible to call this method by passing the proper dialog matcher list. This is generally done in the creation of the new SIP context through the EvOnPacketReceived event or when creating a context to send a new request.

Note that for most cases, applications should support only one instance of the dialog matcher list. The CDialoMatcherList should be created in the initialization process of the application and be shared with every SIP context through this method. Therefore, releasing the CDialoMatcherList should be done in the application finalization. In that scenario, the same instance is passed through this method after the context was created and the required services attached. The dialog matcher list can also be configured in the application initialization through the SetDefaultDialogMatcherList method of the ISipCoreConfig (see page 28).

However, for some reasons, the implementation may wish to support more than one dialog matcher list. In that case, the application is responsible to give the received CSipPacket (see page 447) to the proper dialog matcher list through the EvOnPacketReceived event. As a guide to determine to which dialog the packet must be given, the following is what the application has to look for:

- Request: the source of the packet (both IP and port)
- Response: the topmost Via header (both host and port)

See Also

GetDialogMatcherList (see page 71), CSipDialogMatcherList

3.2.1.7 - ISipForkedDialogGrouper Class

Class Hierarchy

```
ISipForkedDialogGrouper
```

C++

```
class ISipForkedDialogGrouper;
```

Description

This interface regroups multiple dialogs created from a single request that has forked.

The services issuing requests that can create a dialog and that can fork is automatically enabling an object that implements this interface on the request's ISipRequestContext.

When forking happens, the application gets this object through the ISipForkedDialogGrouperMgr::EvNewDialogNeeded (see page 75) method. The manager should then create and configure a new ISipContext (see page 23) and call ContextCreated (see page 73) on this object with that ISipContext (see page 23) and the packet passed in the EvNewDialogNeeded method.

The object is reference counted.

Note that this object is not synchronized and must be accessed only in the context of the core thread.

Location

SipCore/ISipForkedDialogGrouper.h

See Also

ISipForkedDialogGrouperMgr (see page 74)

Methods

Method	Description
◆ A AddRef (see page 73)	Adds a reference on this object.
◆ A ContextCreated (see page 73)	Dispatches the packet resulting from forking to the newly created context.
◆ A ReleaseRef (see page 74)	Releases a reference on this object.

Legend

◆ A	Method
A	abstract

3.2.1.7.1 - Methods

3.2.1.7.1.1 - ISipForkedDialogGrouper::AddRef Method

Adds a reference on this object.

C++

```
virtual unsigned int AddRef() = 0;
```

Returns

The current number of references.

Description

Adds a reference on this object. The added reference must be released in order to let this object destroy itself.

Note that this method is not synchronized and must be accessed only in the context of the core thread.

3.2.1.7.1.2 - ISipForkedDialogGrouper::ContextCreated Method

Dispatches the packet resulting from forking to the newly created context.

C++

```
virtual mxt_result ContextCreated(IN ISipContext& rContext, IN const CSipPacket& rPacket, IN mxt_opaque opqTransaction) = 0;
```

Parameters

Parameters	Description
IN ISipContext& rContext	The newly created context.
IN const CSipPacket& rPacket	The received packet. MUST be the same instance as the one given in EvNewDialogNeeded (i.e. not a copy of the packet).
IN mxt_opaque opqTransaction	Application data to associate with the created transaction. This is opaque to the service.

Returns

- **resS_OK**: The packet is properly processed by the context.
- **resFE_FAIL**: The packet could not be processed by the context. This could happen if the context in parameter is not a newly created context or if the service that originated the forked request is not attached to the context.
- **resFE_DUPLICATE**: Unable to create the server transaction, as it already exists. The context should be released immediately.

Description

This method should be called by the ISipForkedDialogGrouperMgr (see page 74) when EvNewDialogNeeded is called. The manager should create a new ISipContext (see page 23) and attach the appropriate services to it. It should then call this method with that ISipContext (see page 23) and the CSipPacket (see page 447) received in parameter of EvNewDialogNeeded on the ISipForkedDialogGrouper (see page 73) instance received in parameter of EvNewDialogNeeded.

Note that this method is not synchronized and must be accessed only in the context of the core thread.

3.2.1.7.1.3 - ISipForkedDialogGrouper::ReleaseRef Method

Releases a reference on this object.

C++

```
virtual unsigned int ReleaseRef() = 0;
```

Returns

The current number of references.

Description

Releases a reference on this object. When the reference count reaches 0, the object deletes itself.

Note that this method is not synchronized and must be accessed only in the context of the core thread.

3.2.1.8 - ISipForkedDialogGrouperMgr Class**Class Hierarchy**

```
ISipForkedDialogGrouperMgr
```

C++

```
class ISipForkedDialogGrouperMgr;
```

Description

The interface to which the CSipForkedDialogGrouper reports its events.

Location

SipCore/ISipForkedDialogGrouper.h

See Also

CSipForkedDialogGrouper

Methods

Method	Description
◆ A EvNewDialogNeeded (see page 75)	Notifies the manager that a new dialog is needed to handle a new dialog as a result of a sent request that forked.

Legend

◆	Method
A	abstract

3.2.1.8.1 - Methods

3.2.1.8.1.1 - ISipForkedDialogGrouper::EvNewDialogNeeded Method

Notifies the manager that a new dialog is needed to handle a new dialog as a result of a sent request that forked.

C++

```
virtual void EvNewDialogNeeded(IN ISipForkedDialogGrouper& rGrouper, IN const CSipPacket& rPacket, IN ISipContext& rOriginator) = 0;
```

Parameters

Parameters	Description
IN ISipForkedDialogGrouper& rGrouper	The grouper that needs a new dialog (i.e. the one on which ContextCreated must be called).
IN const CSipPacket& rPacket	The received packet that needs to be handled on a new dialog.
IN ISipContext& rOriginator	The context that sent the request that initiated the dialog.

Description

This method notifies the manager that a new dialog is needed because a dialog creating request was forked and the originator request is already associated with another dialog.

The manager that implements this method should create a new context with a configuration similar to the originator ISipContext (see page 23). As a minimum, the service that generated the request that was forked should be added to the new context. Once the context is created, the manager should call ContextCreated on rGrouper.

If the application does not want to allocate resources for this new dialog, it should still create a context and add the appropriate services to reject that dialog. The rejection depends on the type of request that created the dialog. For dialogs created with an INVITE request, the application should send a BYE. For dialogs created with a SUBSCRIBE request, the application could either reject the NOTIFY received with a 481 response or preferably accept the NOTIFY and send a SUBSCRIBE request with an expiration of 0 seconds to terminate the subscription. Note however that terminating supplemental dialogs can lead to interoperation problems since the actual target that you are trying to reach is not necessarily the first one to create a dialog.

This event can be processed synchronously or asynchronously. If the manager processes this event asynchronously, it must make sure to count references for the three objects in parameter.

See Also

ISipForkedDialogGrouper (see page 73), ISipSessionSvc::Bye (see page 580), ISipServerEventCtrl::SendResponse.

3.2.1.9 - ISipServerEventControl Class

Class Hierarchy



C++

```
class ISipServerEventControl : public IEComUnknown;
```

Description

This interface is provided to managers that are receiving stack events associated with a server transaction. It is provided as a parameter to such events.

A final response MUST be sent using this interface or the ISipDialogServerEventControl interface before the context generating the event can be cleared and released.

This interface can be used to send multiple provisional (1xx) responses and only one final response.

This interface is an ECOM interface and as such ECOM reference rules apply.

Location

SipCore/ISipServerEventControl.h

Methods

Method	Description
● A GetOpaque (see page 76)	Gets the user-specified opaque parameter associated with this transaction.
● A SendResponse (see page 76)	Sends a response for a request received on this transaction.
● A SetOpaque (see page 77)	Sets the user-specified opaque parameter associated with this transaction.

Legend

● A	Method
A	abstract

3.2.1.9.1 - Methods**3.2.1.9.1.1 - ISipServerEventControl::GetOpaque Method**

Gets the user-specified opaque parameter associated with this transaction.

C++

```
virtual mxtr_opaque GetOpaque() = 0;
```

Returns

The opaque parameter.

Description

Gets the user-specified opaque parameter associated with this transaction.

This is the opaque parameter that is set through ISipServerEventControl::SetOpaque (see page 77).

See Also

SetOpaque (see page 77)

3.2.1.9.1.2 - ISipServerEventControl::SendResponse Method

Sends a response for a request received on this transaction.

C++

```
virtual mxtr_result SendResponse(IN unsigned int uCode, IN const char* szReason, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody) = 0;
```

Parameters

Parameters	Description
IN unsigned int uCode	The response code to use in the sent response.

IN const char* szReason	The reason phrase to use in the sent response. If NULL, the default reason phrase for the specified uCode is used.
IN TO CHeaderList* pExtraHeaders	Additional headers to put in the sent response. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Message-body to put in the sent response. It can be NULL. Ownership is TAKEN.

Returns

- resS_OK: Response is being sent.
- resFE_INVALID_STATE: The attached ISipUserAgentSvc (see page 639) is incorrectly configured. For instance, the contact list could be empty.
- resFE_FAIL: A final response was already sent on this transaction.
- resFE_FAIL: The response could not be sent.

Description

Creates a response with the described status code and reason phrase for the request received on the server transaction. It then adds the extra headers and content information if any is provided in parameter.

This method can be called as often as required for provisional responses (1xx status code) and MUST be called once for final response (status code >= 200). Once SendResponse has been successfully called to send a final response, the manager should release its reference to this interface.

3.2.1.9.1.3 - ISipServerEventControl::SetOpaque Method

Sets the user-specified opaque parameter associated with this transaction.

C++

```
virtual void SetOpaque(IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opq	The opaque parameter.

Description

Sets the user-specified opaque parameter associated with this transaction.

This is the opaque parameter that is obtained through ISipServerEventControl::GetOpaque (see page 76).

See Also

GetOpaque (see page 76)

3.2.2 - Enumerations

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

3.3 - SipCoreSvc

This section documents the Sources/SipCoreSvc folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 78)
- Enumerations (see page 126)
- Types (see page 126)

3.3.1 - Classes

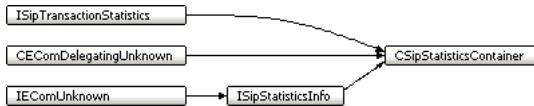
This section documents the classes of the Sources/SipCoreSvc folder.

Classes

Class	Description
CSipStatisticsContainer (See page 78)	
ISipConnectionBlacklistMgr (See page 83)	The connection blacklist manager that informs the application about the insertion or removal of IP addresses from the blacklist.
ISipConnectionBlacklistSvc (See page 84)	
ISipConnectionReuseSvc (See page 85)	
ISipCoreOutputControllingMgr (See page 88)	
ISipCoreOutputControllingSvc (See page 90)	
ISipDestinationSelectionSvc (See page 92)	This service is responsible to set the next-hop URI in the packet.
ISipDiversionSvc (See page 93)	
ISipEnumRequestHandlerMgr (See page 94)	
ISipEnumSvc (See page 95)	
ISipOptionTagsSvc (See page 96)	
ISipOutboundConnectionMgr (See page 100)	
ISipOutboundConnectionSvc (See page 102)	
ISipPersistentConnectionMgr (See page 105)	
ISipPersistentConnectionSvc (See page 107)	
ISipServerLocationSvc (See page 108)	
ISipStatelessDigestServerAuthSvc (See page 110)	
ISipStatisticsInfo (See page 116)	
ISipStatisticsSvc (See page 123)	
ISipSymmetricUdpSvc (See page 124)	
ISipViaManagementSvc (See page 125)	This service is responsible to set the top-most Via header.

3.3.1.1 - CSipStatisticsContainer Class

Class Hierarchy



C++

```
class CSipStatisticsContainer : public ISipStatisticsInfo, public ISipTransactionStatistics, private CEComDelegatingUnknown;
```

Description

This class contains the entire statistics defined in the class ISipStatisticsInfo (See page 116). When passed to the statistics or server location service, data is added to this container by using the implemented interface of ISipTransactionStatistics. Statistics can be retrieved at any time by calling one of the get methods. The same container can be used for all contexts or a new one can be used for each context.

Note that a DNS request done by the CSipUAAssertedIdentitySvc, CSipPersistentConnexionList, or CSipPrivacySvc are not counted in the statistics.

Packets locally generated in the transaction are not counted in the statistics. These packets are not received by a socket. They are only generated to properly terminate the transaction.

The CSipStatisticsContainer is an ECOM object

The CSipStatisticsContainer is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipTransactionStatistics

Interface Id: IID_ISipTransactionStatistics

A user can call QueryIf on a CLSID_CSipStatisticsContainer object created with the CreateEComInstance method to get either the ISipTransactionStatistics or the ISipStatisticsInfo (see page 116) interface of the object.

Location

SipCoreSvc/CSipStatisticsContainer.h

See Also

ISipStatisticsInfo (see page 116), ISipTransactionStatistics

Destructors

Destructor	Description
◆▼ ~CSipStatisticsContainer (see page 80)	Destructor.

Legend

◆	Method
▼	virtual

Methods

Method	Description
◆▼ GetNumActiveTransactions (see page 80)	Gets the total number of active transactions.
◆▼ GetNumDnsQueriesFailed (see page 80)	Gets the number of times a DNS query has failed.
◆▼ GetNumDnsQueriesSucceeded (see page 80)	Gets the number of times a DNS query has succeeded.
◆▼ GetNumFinalResponseRetransmissionsRx (see page 81)	Gets the number of times a final response to the specified method has been received and processed as a retransmission.
◆▼ GetNumFinalResponseRetransmissionsTx (see page 81)	Gets the number of times a final response to the specified method has been retransmitted.
◆▼ GetNumFinalResponseRx (see page 81)	Gets the number of times the specified final response has been received.
◆▼ GetNumFinalResponseTx (see page 81)	Gets the number of times the specified final response has been sent.
◆▼ GetNumNonFinalResponseRetransmissionsTx (see page 81)	Gets the number of times a non-final response to the specified method has been retransmitted.
◆▼ GetNumNonFinalResponseRx (see page 81)	Gets the number of times a non-final response to the specified method has been received.
◆▼ GetNumNonFinalResponseTx (see page 81)	Gets the number of times the specified non-final response has been sent.
◆▼ GetNumRequestRetransmissionsRx (see page 81)	Gets the number of times the specified method has been received and processed as a retransmission.
◆▼ GetNumRequestRetransmissionsTx (see page 81)	Gets the number of times the specified method has been retransmitted.
◆▼ GetNumRequestsRx (see page 82)	Gets the number of times the specified method has been received.
◆▼ GetNumRequestsTx (see page 82)	Gets the number of times the specified method has been sent.
◆▼ GetTotalNumRequestsRx (see page 82)	Gets the total number of requests received.
◆▼ GetTotalNumRequestsTx (see page 82)	Gets the total number of requests sent.
◆▼ GetTotalNumResponsesRx (see page 82)	Gets the total number of responses received.
◆▼ GetTotalNumResponsesTx (see page 82)	Gets the total number of responses sent.
◆▼ GetTotalNumTransactions (see page 82)	Gets the total number of transactions.
◆ NotifyDnsQueryResult (see page 82)	Notifies that a DNS query has been done.
◆ NotifyReceivedPacket (see page 83)	Notifies that a packet has been received.
◆ NotifySentPacket (see page 83)	Notifies that a packet has been successfully sent.
◆ NotifyTransactionEnd (see page 83)	Notifies that a transaction has been terminated.
◆ NotifyTransactionStart (see page 83)	Notifies that a new transaction has been created.
◆▼ Reset (see page 83)	Resets all statistics.

ISipStatisticsInfo Class

ISipStatisticsInfo Class	Description
◆▼ A GetNumActiveTransactions (see page 117)	Gets the total number of active transactions.
◆▼ A GetNumDnsQueriesFailed (see page 117)	Gets the number of times a DNS query has failed.
◆▼ A GetNumDnsQueriesSucceeded (see page 118)	Gets the number of times a DNS query has succeeded.
◆▼ A GetNumFinalResponseRetransmissionsRx (see page 118)	Gets the number of times a final response to the specified SIP method has been received and processed as a retransmission.

• A GetNumFinalResponseRetransmissionsTx (see page 118)	Gets the number of times a final response to the specified SIP method has been retransmitted.
• A GetNumFinalResponseRx (see page 118)	Gets the number of times the specified final response has been received.
• A GetNumFinalResponseTx (see page 119)	Gets the number of times the specified final response has been sent.
• A GetNumNonFinalResponseRetransmissionsTx (see page 119)	Gets the number of times a non-final response to the specified SIP method has been retransmitted.
• A GetNumNonFinalResponseRx (see page 119)	Gets the number of times a non-final response to the specified SIP method has been received.
• A GetNumNonFinalResponseTx (see page 120)	Gets the number of times the specified non-final response has been sent.
• A GetNumRequestRetransmissionsRx (see page 120)	Gets the number of times the specified SIP method has been received and processed as a retransmission.
• A GetNumRequestRetransmissionsTx (see page 120)	Gets the number of times the specified SIP method has been retransmitted.
• A GetNumRequestsRx (see page 121)	Gets the number of times the specified SIP method has been received.
• A GetNumRequestsTx (see page 121)	Gets the number of times the specified SIP method has been sent.
• A GetTotalNumRequestsRx (see page 121)	Gets the total number of requests received.
• A GetTotalNumRequestsTx (see page 121)	Gets the total number of requests sent.
• A GetTotalNumResponsesRx (see page 122)	Gets the total number of responses received.
• A GetTotalNumResponsesTx (see page 122)	Gets the total number of responses sent.
• A GetTotalNumTransactions (see page 122)	Gets the total number of transactions.
• A Reset (see page 122)	Resets all statistics.

Legend

•	Method
V	virtual
A	abstract

3.3.1.1.1 - Destructors

3.3.1.1.1.1 - CSipStatisticsContainer::~CSipStatisticsContainer Destructor

Destructor.

C++

```
virtual ~CSipStatisticsContainer();
```

3.3.1.1.2 - Methods

3.3.1.1.2.1 - CSipStatisticsContainer::GetNumActiveTransactions Method

Gets the total number of active transactions.

C++

```
virtual unsigned int GetNumActiveTransactions() const;
```

3.3.1.1.2.2 - CSipStatisticsContainer::GetNumDnsQueriesFailed Method

Gets the number of times a DNS query has failed.

C++

```
virtual unsigned int GetNumDnsQueriesFailed() const;
```

3.3.1.1.2.3 - CSipStatisticsContainer::GetNumDnsQueriesSucceeded Method

Gets the number of times a DNS query has succeeded.

C++

```
virtual unsigned int GetNumDnsQueriesSucceeded() const;
```

3.3.1.1.2.4 - CSipStatisticsContainer::GetNumFinalResponseRetransmissionsRx Method

Gets the number of times a final response to the specified method has been received and processed as a retransmission.

C++

```
virtual unsigned int GetNumFinalResponseRetransmissionsRx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.5 - CSipStatisticsContainer::GetNumFinalResponseRetransmissionsTx Method

Gets the number of times a final response to the specified method has been retransmitted.

C++

```
virtual unsigned int GetNumFinalResponseRetransmissionsTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.6 - CSipStatisticsContainer::GetNumFinalResponseRx Method

Gets the number of times the specified final response has been received.

C++

```
virtual unsigned int GetNumFinalResponseRx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.7 - CSipStatisticsContainer::GetNumFinalResponseTx Method

Gets the number of times the specified final response has been sent.

C++

```
virtual unsigned int GetNumFinalResponseTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.8 - CSipStatisticsContainer::GetNumNonFinalResponseRetransmissionsTx Method

Gets the number of times a non-final response to the specified method has been retransmitted.

C++

```
virtual unsigned int GetNumNonFinalResponseRetransmissionsTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.9 - CSipStatisticsContainer::GetNumNonFinalResponseRx Method

Gets the number of times a non-final response to the specified method has been received.

C++

```
virtual unsigned int GetNumNonFinalResponseRx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.10 - CSipStatisticsContainer::GetNumNonFinalResponseTx Method

Gets the number of times the specified non-final response has been sent.

C++

```
virtual unsigned int GetNumNonFinalResponseTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.11 - CSipStatisticsContainer::GetNumRequestRetransmissionsRx Method

Gets the number of times the specified method has been received and processed as a retransmission.

C++

```
virtual unsigned int GetNumRequestRetransmissionsRx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.12 - CSipStatisticsContainer::GetNumRequestRetransmissionsTx Method

Gets the number of times the specified method has been retransmitted.

C++

```
virtual unsigned int GetNumRequestRetransmissionsTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.13 - CSipStatisticsContainer::GetNumRequestsRx Method

Gets the number of times the specified method has been received.

C++

```
virtual unsigned int GetNumRequestsRx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.14 - CSipStatisticsContainer::GetNumRequestsTx Method

Gets the number of times the specified method has been sent.

C++

```
virtual unsigned int GetNumRequestsTx(IN const ESipMethod eMethod) const;
```

3.3.1.1.2.15 - CSipStatisticsContainer::GetTotalNumRequestsRx Method

Gets the total number of requests received.

C++

```
virtual unsigned int GetTotalNumRequestsRx() const;
```

3.3.1.1.2.16 - CSipStatisticsContainer::GetTotalNumRequestsTx Method

Gets the total number of requests sent.

C++

```
virtual unsigned int GetTotalNumRequestsTx() const;
```

3.3.1.1.2.17 - CSipStatisticsContainer::GetTotalNumResponsesRx Method

Gets the total number of responses received.

C++

```
virtual unsigned int GetTotalNumResponsesRx() const;
```

3.3.1.1.2.18 - CSipStatisticsContainer::GetTotalNumResponsesTx Method

Gets the total number of responses sent.

C++

```
virtual unsigned int GetTotalNumResponsesTx() const;
```

3.3.1.1.2.19 - CSipStatisticsContainer::GetTotalNumTransactions Method

Gets the total number of transactions.

C++

```
virtual unsigned int GetTotalNumTransactions() const;
```

3.3.1.1.2.20 - CSipStatisticsContainer::NotifyDnsQueryResult Method

Notifies that a DNS query has been done.

C++

```
void NotifyDnsQueryResult(IN bool bRes);
```

3.3.1.1.2.21 - CSipStatisticsContainer::NotifyReceivedPacket Method

Notifies that a packet has been received.

C++

```
void NotifyReceivedPacket(IN const CSipPacket& rPacket, IN bool bRetransmission);
```

3.3.1.1.2.22 - CSipStatisticsContainer::NotifySentPacket Method

Notifies that a packet has been successfully sent.

C++

```
void NotifySentPacket(IN const CSipPacket& rPacket, IN bool bRetransmission);
```

3.3.1.1.2.23 - CSipStatisticsContainer::NotifyTransactionEnd Method

Notifies that a transaction has been terminated.

C++

```
void NotifyTransactionEnd();
```

3.3.1.1.2.24 - CSipStatisticsContainer::NotifyTransactionStart Method

Notifies that a new transaction has been created.

C++

```
void NotifyTransactionStart();
```

3.3.1.1.2.25 - CSipStatisticsContainer::Reset Method

Resets all statistics.

C++

```
virtual void Reset();
```

3.3.1.2 - ISipConnectionBlacklistMgr Class

The connection blacklist manager that informs the application about the insertion or removal of IP addresses from the blacklist.

Class Hierarchy

```
ISipConnectionBlacklistMgr
```

C++

```
class ISipConnectionBlacklistMgr;
```

Description

The connection blacklist manager informs the application when an IP address is added to the blacklist or removed from it. This manager is optional for the application. It can be configured using the SetManager method of CSipConnectionBlacklist.

Location

SipCoreSvc/ISipConnectionBlacklistMgr.h

See Also

ISipCoreConfig (see page 28), CSipConnectionBlacklist

Methods

Method	Description
♪ A EvAddressBlacklisted (see page 84)	Informs that a given address has been inserted into the blacklist.
♪ A EvBlacklistDurationCompleted (see page 84)	Invoked when the address has completed its penalty in the blacklist.

Legend

	Method
	abstract

3.3.1.2.1 - Methods**3.3.1.2.1.1 - ISipConnectionBlacklistMgr::EvAddressBlacklisted Method**

Informs that a given address has been inserted into the blacklist.

C++

```
virtual void EvAddressBlacklisted(IN const CSocketAddr& rPeerAddr, IN ESipTransport eTransport) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddr	The address of the remote peer that was blacklisted.
IN ESipTransport eTransport	The transport protocol for the blacklisted address.

Description

This event is invoked when a new address for a given transport protocol is inserted into the blacklist. It can be used by the application to update its subscriptions for example.

Notes

The rPeerAddr parameter is only valid for the duration of this method.

3.3.1.2.1.2 - ISipConnectionBlacklistMgr::EvBlacklistDurationCompleted Method

Invoked when the address has completed its penalty in the blacklist.

C++

```
virtual void EvBlacklistDurationCompleted(IN const CSocketAddr& rPeerAddr, IN ESipTransport eTransport) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddr	The address of the remote peer that was blacklisted.
IN ESipTransport eTransport	The transport protocol for the blacklisted address.

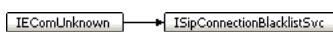
Description

This event is invoked when the address has completed its penalty time in the blacklist. This event can occur either when the address was removed from the blacklist or when the duration of the penalty timer has elapsed.

Such an event can be useful to the application to determine when a blacklisted IP address has become available. That means the SIP stack will try to reuse this address when a packet needs to be sent to the host that maps with this IP.

Notes

The rPeerAddr parameter is only valid for the duration of this method.

3.3.1.3 - ISipConnectionBlacklistSvc Class**Class Hierarchy****C++**

```
class ISipConnectionBlacklistSvc : public IEComUnknown;
```

Description

This service automatically manages a blacklist of destinations to use. A destination is an IP address, a port number, and a transport. A blacklist is a list of destinations that are not used to contact a peer. The only way a destination blacklisted is used is when all possible destinations to reach a peer are blacklisted. In this case, the preferred one is used.

When sending a packet to a destination fails, this service adds the destination to the blacklist for the configured amount of time. After the blacklisted destination expires, it is removed from the blacklist.

When sending a packet to a destination succeeds, this service removes the destination from the blacklist.

The blacklist can directly be accessed by the application. To access it, call CSipConnectionBlacklist::Instance().

IMPORTANT: To work properly, this service must be added to the context AFTER the ISipServerLocationSvc (see page 108).

The ISipConnectionBlacklistSvc is an ECOM object

The ISipConnectionBlacklistSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipConnectionBlacklistSvc

Interface Id: IID_ISipConnectionBlacklistSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipConnectionBlacklistSvc.h

See Also

CSipConnectionBlacklist

3.3.1.4 - ISipConnectionReuseSvc Class

Class Hierarchy



C++

```
class ISipConnectionReuseSvc : public IEComUnknown;
```

Description

This connection service allows the application to reuse a connected socket for sending requests that was created from an incoming request. This service is used to limit the creation of new connections to the peer that sent a request.

This is to reuse the initial TLS connection created by a peer when sending requests inside the same context. The service sends the request to the address of the last received request. If this service is not used and the contact info of the peer is not set properly, the SIP stack will create a new connection to the peer.

This service MUST NOT be used with the ISipServerLocationSvc (see page 108). The ISipServerLocationSvc (see page 108) will effectively override what this service does. It will also conflict with the ISipPersistentConnectionSvc (see page 107). The last service added will have precedence over the first one as they both manipulate the same data.

Please note that if this service is attached, there will be no fallback tried by the SIP stack.

Another use of this service can be to set the local and peer addresses to be used for sending outgoing requests from this service using the SetLocalAddress (see page 87) and the SetPeerAddress (see page 87) methods. These can be retrieved from another such service using the GetLocalAddress (see page 86) and GetPeerAddress (see page 86) methods.

The ISipConnectionReuseSvc is an ECOM object

The ISipConnectionReuseSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipConnectionReuseSvc

Interface Id: IID_ISipConnectionReuseSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipConnectionReuseSvc.h

Methods

Method	Description
◆ A GetLocalAddress (see page 86)	Gets the local address.
◆ A GetPeerAddress (see page 86)	Gets the peer address.
◆ A GetTransport (see page 86)	Gets the transport.
◆ A SetLocalAddress (see page 87)	Sets the local address.
◆ A SetPeerAddress (see page 87)	Sets the peer address.
◆ A SetTransport (see page 87)	Sets the transport.

Legend

◆	Method
A	abstract

3.3.1.4.1 - Methods**3.3.1.4.1.1 - ISipConnectionReuseSvc::GetLocalAddress Method**

Gets the local address.

C++

```
virtual const CSocketAddr& GetLocalAddress() = 0;
```

Returns

The local address.

Description

Gets the local address to be used to send a request to a peer.

3.3.1.4.1.2 - ISipConnectionReuseSvc::GetPeerAddress Method

Gets the peer address.

C++

```
virtual const CSocketAddr& GetPeerAddress() = 0;
```

Returns

The peer address.

Description

Gets the peer address to be used to send a request to a peer.

3.3.1.4.1.3 - ISipConnectionReuseSvc::GetTransport Method

Gets the transport.

C++

```
virtual ESipTransport GetTransport() = 0;
```

Parameters

Parameters	Description
eTransport	The transport.

Description

Gets the transport for this service.

3.3.1.4.1.4 - ISipConnectionReuseSvc::SetLocalAddress Method

Sets the local address.

C++

```
virtual void SetLocalAddress(IN const CSocketAddr& rLocalAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rLocalAddress	The local address.

Description

Sets the local address to be used to send a request to a peer.

NOTES: Both the peer and the local address must be set to a valid value and the transport to modify the outgoing packet.

3.3.1.4.1.5 - ISipConnectionReuseSvc::SetPeerAddress Method

Sets the peer address.

C++

```
virtual void SetPeerAddress(IN const CSocketAddr& rPeerAddress) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rPeerAddress	The peer address.

Description

Sets the peer address to be used to send a request to a peer.

NOTES: Both the peer and the local address must be set to a valid value and the transport to modify the outgoing packet.

3.3.1.4.1.6 - ISipConnectionReuseSvc::SetTransport Method

Sets the transport.

C++

```
virtual void SetTransport(IN ESipTransport eTransport) = 0;
```

Parameters

Parameters	Description
IN ESipTransport eTransport	The transport.

Description

Sets the transport for this service to use.

NOTES: Both the peer and the local address must be set to a valid value and the transport to modify the outgoing packet.

3.3.1.5 - ISipCoreOutputControllingMgr Class

Class Hierarchy

ISipCoreOutputControllingMgr

C++

```
class ISipCoreOutputControllingMgr;
```

Description

This interface is used by the output controlling service (ISipCoreOutputControllingSvc (see page 90)) when asked by the context to update a packet.

This interface is used to notify the application that a packet should be updated. This happens when the output controlling service sees its turn come up in the service list attached to the context. The update may be related to the network interface to use when sending the packet. It may also be related to some headers.

Location

SipCoreSvc/ISipCoreOutputControllingMgr.h

See Also

ISipCoreOutputControllingSvc (see page 90), CSipPacket (see page 447)

Methods

Method	Description
EvUpdatePacket (see page 88)	Notifies the application that a packet should be updated.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
EUpdatingSynchronization (see page 90)	

3.3.1.5.1 - Methods

3.3.1.5.1.1 - ISipCoreOutputControllingMgr::EvUpdatePacket Method

Notifies the application that a packet should be updated.

C++

```
virtual void EvUpdatePacket(IN ISipCoreOutputControllingSvc* pSvc, IN mxt_opaque transactionOpaque, IN
mxt_opaque opqId, INOUT CSipPacket& rPacket, OUT EUpdatingSynchronization& reSynchronization, OUT mxt_result&
rresUpdate) = 0;
```

Parameters

Parameters	Description
IN ISipCoreOutputControllingSvc* pSvc	The service currently managing the packet update.
IN mxt_opaque transactionOpaque	The opaque parameter associated with the transaction.
IN mxt_opaque opqId	An opaque value associated with the current updated packet. This value must be passed to the ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated (see page 91) method when the packet is updated asynchronously. It is mandatory that the SAME opaque value be passed along the corresponding updated packet.

INOUT CSipPacket& rPacket	The packet to update. If the updating is done asynchronously, the SAME packet (not a copy) must be passed to ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated (see page 91) with its corresponding opaque value. The packet is only guaranteed to be valid for the duration of the call. If the manager needs it for longer, for instance when doing asynchronous updates, a reference must be obtained and later released.
OUT EUpdatingSynchronization& reSynchronization	This parameter must be set by the manager. It indicates to the calling service whether the update will be completed at a later time.
OUT mxt_result& rresUpdate	This parameter must be set by the manager if reSynchronization is set to eSYNCHRONOUSLY_UPDATED. It indicates if the packet update was successful. This parameter is ignored if the update is asynchronous. The possible results are as follows:
eSYNCHRONOUSLY_UPDATED	The packet is updated or will not be updated at all (synchronous update). In this case, there is no need to call PacketAsynchronouslyUpdated on the service (doing so would fail).
eASYNCHRONOUSLY_UPDATED	The packet is not yet updated but will be at a later time (asynchronous update). In this case, once the packet is updated, PacketAsynchronouslyUpdated MUST be called on the service. This must be done with the same packet and opaque value (opId) as passed to this method. Note that until PacketUpdated is called, no processing is done on the packet by the M5T stack.
ress_OK	The update has been successful and the packet should be sent on the network.
resFE_FAIL	The update has failed for some undisclosed reason. For instance, it could be because there are no routes for the packet source and destination. In this case, the packet is not sent on the network. If it is a response, it is dropped. If it is a request, a 503 Service Unavailable final negative response is automatically generated.

Description

Notifies the application that an outgoing packet should be updated.

If the update is done synchronously, there is nothing special to do other than set the reSynchronization parameter to eSYNCHRONOUSLY_UPDATED and return.

If the manager does not want to update the packet, it should do nothing but set reSynchronization to eSYNCHRONOUSLY_UPDATED.

If the update is done asynchronously, the manager must set the reSynchronization parameter to eASYNCHRONOUSLY_UPDATED before returning from this call. Then, after the packet is updated, the manager must call PacketAsynchronouslyUpdated on the service. It is mandatory that the SAME opaque ID (opId) and packet be passed back to the service in the call to ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated (see page 91).

Note that when such an asynchronous update occurs, internal processing of the packet is interrupted. For instance, the associated context stops forwarding the packet to downstream services, etc. Consequently, each call to this method which returns eASYNCHRONOUSLY_UPDATED MUST be matched to a call to PacketAsynchronouslyUpdated on the service.

One important caveat of the output controlling service is when multiple packets are being processed asynchronously by the application. In this case, it is possible for the application to effectively change the order packets are sent onto the network by calling PacketAsynchronouslyUpdated on out-of-order packets. Typically, this should not occur and an application should only do so very cautiously.

Indeed, if packets are sent out of order the CSeq header should be directly massaged by the manager. This is particularly true when the requests are sent in a dialog or if the CSeq value has a special meaning even outside a dialog (e.g., REGISTER). In these cases, if the requests are sent out of order with unchanged CSeq headers, the server will refuse the requests with a lower CSeq than the last request received. Note that this can also happen when mixing synchronous and asynchronous updates. In this case, a packet updated synchronously could be sent to the network before a packet being asynchronously updated. Generally, if possible, it is simpler for an application to update packets sequentially and not mix synchronous and asynchronous processing. A good way to avoid disordering packets would be to queue them in a FIFO queue, along with their corresponding opaque ID.

The rPacket parameter is only guaranteed to be valid for the duration of the call. Thus, for asynchronous updates, a manager will add a reference to the packet before returning. Later, once the packet is updated, the manager will call PacketAsynchronouslyUpdated on the service and then release its reference on the packet.

Possible usage of the output controlling service includes verifying if the network interface chosen to send the packet is the one the application would also choose. If not, it could do some modifications to the packet such as changing the listening address and port for RTP and the address in the Via, From, and Contact headers. It could also update the corresponding configuration in the ISipUserAgentSvc (see page 639) of the context.

Care should be taken while updating any of the headers. Some headers mean different things depending in which request or response they are. For example, the Contact header has a special meaning in REGISTER requests and responses.

See Also

[ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated](#) (see page 91), [CSipPacket](#) (see page 447), [CSipPacket::GetLocalAddr](#) (see page 452)

3.3.1.5.2 - Enumerations**3.3.1.5.2.1 - ISipCoreOutputControllingMgr::EUpgradingSynchronization Enumeration**

```
enum EUpgradingSynchronization {
    eSYNCHRONOUSLY_UPDATED,
    eASYNCHRONOUSLY_UPDATED
};
```

Description

Indicates how the packet is updated.

Members

Members	Description
eSYNCHRONOUSLY_UPDATED	The packet is updated or will not be updated at all.
eASYNCHRONOUSLY_UPDATED	The packet is not yet updated but will be at a later time.

3.3.1.6 - ISipCoreOutputControllingSvc Class**Class Hierarchy****C++**

```
class ISipCoreOutputControllingSvc : public IEComUnknown;
```

Description

The output controlling service is a connection service that notifies its manager when asked to update a packet that is about to be sent. The manager can then update the packet in any way it sees fit.

The order in which this service is attached to the context is important. This will decide at what moment the service will consult its manager. For example, if the ISipCoreOutputControllingSvc is attached after all the connection services, the manager will see the packet after all the connection services have modified it.

The ISipCoreOutputControllingSvc is an ECOM object

The ISipCoreOutputControllingSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipCoreOutputControllingSvc

Interface Id: IID_ISipCoreOutputControllingSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipCoreOutputControllingSvc.h

See Also

[ISipCoreOutputControllingMgr](#) (see page 88)

Methods

Method	Description
PacketAsynchronouslyUpdated (see page 91)	Notifies this service that a packet is ready to be sent after an asynchronous update by the application.

 SetManager (see page 91)

Configures the manager associated with this service.

Legend

	Method
	abstract

3.3.1.6.1 - Methods

3.3.1.6.1.1 - ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated Method

Notifies this service that a packet is ready to be sent after an asynchronous update by the application.

C++

```
virtual mxt_result PacketAsynchronouslyUpdated(IN mxt_opaque opqId, IN TO CSipPacket* pPacket, IN mxt_result resUpdate) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqId	The opaque value received in ISipCoreOutputControllingMgr::EvUpdatePacket (see page 88) as the opqId parameter. This parameter MUST match a previous opaque ID given to the manager.
IN TO CSipPacket* pPacket	A pointer to the updated packet. It MUST point to the SAME packet (not a copy) that was passed in ISipCoreOutputControllingMgr::EvUpdatePacket (see page 88) with the SAME corresponding opaque value. After this method is called, the pointer must no longer be used.
IN mxt_result resUpdate	Result of the packet update. Possible values are as follows:
resS_OK	The update has been successful and the the packet should be sent on the network.
resFE_FAIL	The update has failed for some undisclosed reason. For instance, it could be because there are no routes for the packet source and destination. In this case, the packet is not sent on the network. If it is a response, it is dropped. If it is a request, a 503 Service Unavailable final negative response is automatically generated.

Returns

resFE_INVALID_STATE: Invalid packet. EvUpdatePacket was not called with this packet or the packet was already updated synchronously.

resFE_INVALID_ARGUMENT :

- pPacket is NULL.
- opqId is 0. The opaque ID MUST BE THE SAME as was passed to the manager via EvUpdatePacket.

resS_OK : The updated packet is accepted by this service and will continue being processed by the context.

Description

Notifies this service that a packet is ready to be sent after an asynchronous update by the application. For instance, the application could have chosen an interface on which the packet should be sent. It could also have updated the payload of the packet or fixed its RTP port. Moreover, it could have updated the From, Contact, and Via headers of the packet and updated the ISipUserAgentSvc (see page 639) configuration of these headers if required.

This method must be called successfully for every call to ISipCoreOutputControllingMgr::EvUpdatePacket (see page 88) in which the manager returns ISipCoreOutputControllingMgr::eASYNCHRONOUSLY_UPDATED.

See Also

ISipCoreOutputControllingMgr::EvUpdatePacket (see page 88)

3.3.1.6.1.2 - ISipCoreOutputControllingSvc::SetManager Method

Configures the manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipCoreOutputControllingMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipCoreOutputControllingMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT: pMgr is NULL.

resS_OK: Otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that a manager MUST be associated with this service before it is used. If it has no configured managers, this service will block every outgoing packet.

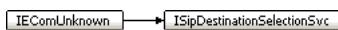
See Also

ISipCoreOutputControllingMgr (see page 88)

3.3.1.7 - ISipDestinationSelectionSvc Class New in 4.1.6

This service is responsible to set the next-hop URI in the packet.

Class Hierarchy



C++

```
class ISipDestinationSelectionSvc : public IEComUnknown;
```

Description

This service sets the next-hop URI, by either using the Request-URI or the URI in the Route header.

Notes

it is mandatory to attach this service on a context and to attach it prior to attaching the server location service (if applicable).

The ISipDestinationSelectionSvc is an ECOM object

The ISipDestinationSelectionSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipDestinationSelectionSvc

Interface Id: IID_ISipDestinationSelectionSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipDestinationSelectionSvc.h

Methods

Method	Description
ForceDestination (see page 93)	Forces a destination to be used.

Legend

	Method
	abstract

3.3.1.7.1 - Methods

3.3.1.7.1.1 - ISipDestinationSelectionSvc::ForceDestination Method

Forces a destination to be used.

C++

```
virtual void ForceDestination(IN const CSipUri& rForcedDestinationUri) = 0;
```

Parameters

Parameters	Description
rDestinationUri	The destination.

Description

Forces the context to use the specified destination when sending requests. The forced URI bypasses the next hop URI specified in the top Route or in the Request-Line (if there is no Route). The SIP protocol specifies that such Route or Request-Line must be used as the next target for requests.

The forced destination will be used as the next hop uri. This means that TLS sockets will use this URI to compare against the common name (CN) of the TLS certificate when validating the handshake.

Warning

This method should be used with care as it breaks the SIP standards.

3.3.1.8 - ISipDiversionSvc Class

Class Hierarchy



C++

```
class ISipDiversionSvc : public IEComUnknown;
```

Description

The diversion service when attached to a UAC is used to copy Diversion headers present in a 3xx to the following INVITEs. Note that the Diversion headers will be copied until a final success response is received. No method on this service needs to be called for this behaviour, all that is needed is to attach the service to a context.

When attached to a UAS the diversion service can be used to automatically add a Diversion header to every outgoing 3xx. To do so simply call the SetReason (see page 94) method with a not NULL reason. To stop adding the header call SetReason (see page 94) with NULL.

This service is the implementation of draft-levy-sip-diversion-08 for the UAC side.

The ISipDiversionSvc is an ECOM object

The ISipDiversionSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipDiversionSvc

Interface Id: IID_ISipDiversionSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipDiversionSvc.h

Methods

Method	Description
SetReason (see page 94)	Sets the reason associated with the diversion.

Legend

	Method
	abstract

3.3.1.8.1 - Methods**3.3.1.8.1.1 - ISipDiversionSvc::SetReason Method**

Sets the reason associated with the diversion.

C++

```
virtual void SetReason(IN const char* pszReason) = 0;
```

Parameters

Parameters	Description
IN const char* pszReason	The reason to set in the Diversion header. If NULL the service stops adding Diversion header to outgoing 3xx.

Description

Sets the reason associated with the diversion. This sets the value of the reason parameter found in a Diversion header. After Calling this method a Diversion header is added to every outgoing 3xx.

3.3.1.9 - ISipEnumRequestHandlerMgr Class**Class Hierarchy**

ISipEnumRequestHandlerMgr

C++

```
class ISipEnumRequestHandlerMgr;
```

Description

This interface is used by the CSipEnumRequestHandler to notify the calling object that the asynchronous query is complete.

Location

SipCoreSvc/ISipEnumRequestHandlerMgr.h

See Also

GetEnumUri, CSipEnumRequestHandler, CSipEnumSvc

Methods

Method	Description
  OnGetEnumUriAResult (see page 94)	Reports the GetEnumUriA result.

Legend

	Method
	abstract

3.3.1.9.1 - Methods**3.3.1.9.1.1 - ISipEnumRequestHandlerMgr::OnGetEnumUriAResult Method**

Reports the GetEnumUriA result.

C++

```
virtual void OnGetEnumUriAResult(IN TO CList<SEnumUri>* plstEnumUri, IN mxt_opaque opq, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN TO CList<SEnumUri>* plstEnumUri	A pointer to a CList of SEnumUri structures containing the URIs that correspond to the supplied tel-URI. Ownership is taken.
IN mxt_opaque opq	A mxt_opaque, given previously to the GetEnumUriA method of CSipEnumRequestHandler.
IN mxt_result res	The result of the GetEnumUri function.

Description

Notifies the manager that the query is complete. The ownership of the list is given back to the manager.

See Also

CSipEnumRequestHandler, GetEnumUri

3.3.1.10 - ISipEnumSvc Class

Class Hierarchy



C++

```
class ISipEnumSvc : public IEComUnknown;
```

Description

This service is responsible to replace the tel-URI of outgoing requests with the sip-URI resulting of an ENUM request as per RFC 3761.

The ISipEnumSvc is an ECOM object

The ISipEnumSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipEnumSvc

Interface Id: IID_ISipEnumSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipEnumSvc.h

Methods

Method	Description
SetServiceBehaviour (see page 95)	Configures the service behaviour.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
EEnumServiceBehaviour (see page 96)	

3.3.1.10.1 - Methods

3.3.1.10.1.1 - ISipEnumSvc::SetServiceBehaviour Method

Configures the service behaviour.

C++

```
virtual void SetServiceBehaviour(IN int nEnumSvcBehaviourBitset) = 0;
```

Parameters

Parameters	Description
IN int nEnumSvcBehaviourBitset	The ENUM service behaviour. This value can be a concatenation of various values created by using the bitwise "OR" operator on the EEnumServiceBehaviour (see page 96) enumeration.

Description

Configures how the ENUM service behaves. The following options may be selected:

- eREMOVE_ROUTE: Removes the Route header after having replaced the request-URI.
- eUPDATE_TO_HEADER: Updates the To header with the resulting ENUM sip-URI.

By default, the above options are unselected.

3.3.1.10.2 - Enumerations**3.3.1.10.2.1 - ISipEnumSvc::EEnumServiceBehaviour Enumeration**

```
enum EEnumServiceBehaviour {
    eNONE = 0,
    eREMOVE_ROUTE = 1,
    eUPDATE_TO_HEADER = 2
};
```

Description

Contains the behaviour option of the ENUM service. The values are bitmaskable except for eNONE.

See Also

SetServiceBehaviour (see page 95).

Members

Members	Description
eNONE = 0	None of the following option.
eREMOVE_ROUTE = 1	Removes the Route header after having replaced the request-URI.
eUPDATE_TO_HEADER = 2	Updates the To header with the resulting ENUM sip-URI.

3.3.1.11 - ISipOptionTagsSvc Class New in 4.1.8**Class Hierarchy****C++**

```
class ISipOptionTagsSvc : public IEComUnknown;
```

Description

The option tags service when attached to a ISipContext (see page 23) is used to automatically add the Supported headers in outgoing requests and 2xx responses. Also, it handles the incoming requests Require headers and automatically answers a 420 Bad Extension if a Required tags is not currently supported.

The ISipOptionTagsSvc is an ECOM object

The ISipOptionTagsSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipOptionTagsSvc

Interface Id: IID_ISipOptionTagsSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipOptionTagsSvc.h

Methods

Method	Description
• A AddSupportedExtension (see page 97)	Adds a custom supported extension.
• A RemoveSupportedExtension (see page 97)	Removes a custom supported extension.
• A SetCustomSupportedExtensions (see page 98)	Sets custom supported extensions.
• A SetManager (see page 98)	Sets the manager associated with the service.
• A SetRequireHeaderVerification (see page 99)	Sets the require header verification behavior.
• A SetSupportedHeaderAddition (see page 99)	Sets the outgoing supported header addition behavior.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
EBehavior (see page 99)	Enumeration defining the incoming and outgoing behavior of the service about which packets to handle.

3.3.1.11.1 - Methods**3.3.1.11.1.1 - ISipOptionTagsSvc::AddSupportedExtension Method**

Adds a custom supported extension.

C++

```
virtual mxt_result AddSupportedExtension(IN const CString& rstrExtension) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrExtension	The custom extension to add.

Returns

resS_OK: The extension was added. resFE_FAIL: The extension was already present.

Description

Adds a custom supported extension to the list.

Warning:

When a custom extension is set, the service does not use the configured global extension list. If the service must also support them, they must be set, one after the others, using this method.

See Also

RemoveSupportedExtension (see page 97)

3.3.1.11.1.2 - ISipOptionTagsSvc::RemoveSupportedExtension Method

Removes a custom supported extension.

C++

```
virtual mxt_result RemoveSupportedExtension(IN const CString& rstrExtension) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrExtension	The custom extension to remove.

Returns

resS_OK: The extension was removed. resFE_FAIL: The extension was not found.

Description

Removes a custom supported extension from the list. To remove all the custom extensions at the same time, use SetCustomSupportedExtensions (see page 98).

Warning:

When one or more custom extension is set, the service does not use the configured global extension list. If the service must also support them, they must be set, one after the others, using this AddSupportedExtension (see page 97).

See Also

SetCustomSupportedExtensions (see page 98), AddSupportedExtension (see page 97)

3.3.1.11.1.3 - ISipOptionTagsSvc::SetCustomSupportedExtensions Method

Sets custom supported extensions.

C++

```
virtual void SetCustomSupportedExtensions(IN TOA CList<CString>* plststrCustomExtensions) = 0;
```

Parameters

Parameters	Description
IN TOA CList<CString>* plststrCustomExtensions	The custom extensions to set on this service. Can be NULL, in which case the default extensions set on the ISipCoreConfig (see page 28) interface will be used.

Description

Sets custom supported extensions. This will replace the old custom extension list if any was set.

Warning:

When a custom extension list is set, the service does not use the configured global extension list. If the service must also support them, they must be within the list or set, one after the others, using AddSupportedExtension (see page 97).

See Also

AddSupportedExtension (see page 97), RemoveSupportedExtension (see page 97)

3.3.1.11.1.4 - ISipOptionTagsSvc::SetManager Method

Sets the manager associated with the service.

C++

```
virtual void SetManager(IN ISipOptionTagsMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipOptionTagsMgr* pMgr	The manager to set. If NULL, the service will still take actions, but will not report them.

Description

Sets the manager associated with the service.

3.3.1.11.1.5 - ISipOptionTagsSvc::SetRequireHeaderVerification Method

Sets the require header verification behavior.

C++

```
virtual void SetRequireHeaderVerification(IN EBehavior eIncomingBehavior) = 0;
```

Parameters

Parameters	Description
IN EBehavior eIncomingBehavior	The behavior to set. This is an incoming behavior.

Description

Sets the incoming require header verification behavior. This will tell the service in which incoming packets to verify the require headers against the local supported extensions.

Default is eBEHAVIOR_HANDLE_REQUESTS_ONLY.

3.3.1.11.1.6 - ISipOptionTagsSvc::SetSupportedHeaderAddition Method

Sets the outgoing supported header addition behavior.

C++

```
virtual void SetSupportedHeaderAddition(IN EBehavior eOutgoingBehavior) = 0;
```

Parameters

Parameters	Description
IN EBehavior eOutgoingBehavior	The behavior to set. This is an outgoing behavior.

Description

Sets the outgoing supported header addition behavior. This will tell the service in which outgoing packets to add the supported headers.

Default is eBEHAVIOR_HANDLE_REQUESTS_AND_2XX_RESPONSES.

3.3.1.11.2 - Enumerations**3.3.1.11.2.1 - ISipOptionTagsSvc::EBehavior Enumeration**

Enumeration defining the incoming and outgoing behavior of the service about which packets to handle.

C++

```
enum EBehavior {
    eBEHAVIOR_HANDLE_NO_PACKET,
    eBEHAVIOR_HANDLE_REQUESTS_ONLY,
    eBEHAVIOR_HANDLE_REQUESTS_AND_2XX_RESPONSES
};
```

Members

Members	Description
eBEHAVIOR_HANDLE_NO_PACKET	No packet will be checked, effectively disabling the service. Valid for both incoming and outgoing behaviors.
eBEHAVIOR_HANDLE_REQUESTS_ONLY	Only requests will be checked. Valid for both incoming and outgoing behaviors.
eBEHAVIOR_HANDLE_REQUESTS_AND_2XX_RESPONSES	Requests and 2xx responses will be checked. Only valid for an outgoing behavior. If set for an incoming behavior, it will have the same behavior as eBEHAVIOR_HANDLE_REQUESTS_ONLY.

3.3.1.12 - ISipOutboundConnectionMgr Class **New in 4.1.6**

Class Hierarchy

```
ISipOutboundConnectionMgr
```

C++

```
class ISipOutboundConnectionMgr;
```

Description

This is the outbound connection manager. It is used by the stack to inform the application when another peer requires the outbound extension. It reports REGISTER responses that contain the Require header with the 'outbound' value. It is also used by the stack to request the application to create a persistent connection.

Location

SipCoreSvc/ISipOutboundConnectionMgr.h

See Also

ISipOutboundConnectionSvc (see page 102)

Methods

Method	Description
◆ A EvConnectionNeeded (see page 100)	Tells that the stack requests a flow to be created.
◆ A EvOutboundRequiredByPeer (see page 101)	Tells the application that the peer supports outbound.

Legend

◆	Method
A	abstract

Enumerations

Enumeration	Description
EConnectionCreation (see page 101)	Offers the flexibility to create or not a persistent connection.

3.3.1.12.1 - Methods

3.3.1.12.1.1 - ISipOutboundConnectionMgr::EvConnectionNeeded Method

Tells that the stack requests a flow to be created.

C++

```
virtual void EvConnectionNeeded(IN ISipOutboundConnectionSvc* pSvc, IN mxt_opaque opqId, IN const CSipPacket& rRequest, IN ESipTransport eTransport, IN const CSocketAddr& rLocalAddr, IN const CSocketAddr& rPeerAddr, OUT EConnectionCreation& reConnectionCreation) = 0;
```

Parameters

Parameters	Description
IN ISipOutboundConnectionSvc* pSvc	The outbound connection service.
IN mxt_opaque opqId	An opaque value that must be passed to the ISipOutboundConnectionSvc::ConnectionCreationResult (see page 103) method when a connection is asynchronously created. It is mandatory that the SAME opaque value is passed with the corresponding connection creation.
IN const CSipPacket& rRequest	The request that triggered the persistent connection creation.
IN ESipTransport eTransport	The transport of the connection.
IN const CSocketAddr& rLocalAddr	The local address of the connection.
IN const CSocketAddr& rPeerAddr	The peer address of the connection.
OUT EConnectionCreation& reConnectionCreation	eCREATE_CONNECTION_ASYNCHRONOUSLY: Must be set to this value if the application wants to create a flow. When the application uses this value, it must not call the method ConnectionCreationResult on the outbound service.

eDO_NOT_CREATE_CONNECTION	Must be set to this value if this application does not want to create a flow. In this case, the request fails to be sent.
---------------------------	---

Description

Informs the application that a flow related to outbound needs to be created. It is possible that some flows are created with the assistance of the application when issuing an outbound request for which there is no persistent connection. This is to be compliant as per draft-ietf-sip-outbound-15 section 4.3 when sending non-REGISTER requests.

This event can be called only when the outbound service is attached after the server location service. It can happen when an application sends a request other than REGISTER for which there is no existing flow. The application can decide when this event is reported to create or not a connection. The connection should be created using the CSipPersistentConnectionList::Establish method.

Before calling the ConnectionCreationResult method on the outbound service, the application must wait for the EvEstablished or EvErrorOnConnection event on the ISipPersistentConnectionList.

See Also

ISipOutboundConnectionSvc::ConnectionCreationResult (see page 103)

3.3.1.12.1.2 - ISipOutboundConnectionMgr::EvOutboundRequiredByPeer Method

Tells the application that the peer supports outbound.

C++

```
virtual void EvOutboundRequiredByPeer(IN ISipOutboundConnectionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket, IN ESipTransport eTransport, IN const CSocketAddr& rLocalAddr, IN const CSocketAddr& rPeerAddr, IN unsigned int uFlowTimer) = 0;
```

Parameters

Parameters	Description
IN ISipOutboundConnectionSvc* pSvc	The outbound connection service.
IN ISipClientEventControl* pClientEventCtrl	The interface to the client event control for this transaction.
IN const CSipPacket& rPacket	The REGISTER response containing the Require:outbound header.
IN ESipTransport eTransport	The transport of the connection.
IN const CSocketAddr& rLocalAddr	The local address of the connection.
IN const CSocketAddr& rPeerAddr	The peer address of the connection.
IN unsigned int uFlowTimer	The value of the Flow-Timer header. Can be zero if not present in the REGISTER response.

Description

Informs the application that a REGISTER 2xx response containing the Require header with 'outbound' value has been received. This usually means that the application can start sending keep alives to the peer when this event is reported. Depending on which transport the flow is for, the application can either start the CRLF or STUN keep alives.

CRLF keep alives can be started by the application using the StartKeepAlive method on the persistent connection list. The stack does not do any STUN keep alive on itself, it is the responsibility of the application. As such, the application can use the Send method on the persistent connection list and the ISipKeepAliveExtensionMgr.

See Also

CSipPersistentConnectionList::StartKeepAlive, CSipPersistentConnectionList::Send, ISipKeepAliveExtensionMgr

3.3.1.12.2 - Enumerations

3.3.1.12.2.1 - ISipOutboundConnectionMgr::EConnectionCreation Enumeration

Offers the flexibility to create or not a persistent connection.

C++

```
enum EConnectionCreation {
    eCREATE_CONNECTION_ASYNCHRONOUSLY,
    eDO_NOT_CREATE_CONNECTION
}
```

};

Members

Members	Description
eCREATE_CONNECTION_ASYNCHRONOUSLY	Used when the application wants to create a flow asynchronously.
eDO_NOT_CREATE_CONNECTION	Used when the application does not want to create a flow.

3.3.1.13 - ISipOutboundConnectionSvc Class New in 4.1.6

Class Hierarchy



C++

```
class ISipOutboundConnectionSvc : public IEComUnknown;
```

Description

This connection service allows the application to manage outbound persistent connections (flows) when acting as a user-agent entity. The service selects the flow to use when sending a request and updates the SIP packets as described in the draft-ietf-sip-outbound-15. The application can use this service to associate flows with a context. It is possible to hold multiple flows to the same domain. As such, the service allows the application to specify which flow must be used when sending a request to a domain name for which there are more than one flow.

The responsibilities of the service are:

- Add the instance-id to REGISTER Contact headers.
- Add the reg-id to REGISTER Contact headers.
- Add the 'ob' parameter to dialog forming requests such as INVITE and SUBSCRIBE.
- Add the 'path' and 'outbound' option-tags in Supported headers.
- Tell the application when outbound is supported by the peer.
- Allow failover using another outbound connection.

Outbound mechanism offers redundancy that occurs when a user registers to multiple proxies within the same domain. As an example, let's pretend that there are three proxies within the domain a.com: reg1.a.com, reg2.a.com and reg3.a.com. A user-agent registers to the three proxies through a persistent connection established to each proxy. The address-of-record (AOR) of the user is the same on all three proxies. When the user wants to place an outgoing call or send a request, it sends the request to a.com. The request could be routed to one of the three proxies as they all serve the same domain, which is a.com. If an outbound connection to one of the Proxies is down, then the next available connection could be used.

The outbound service can be attached AFTER the server location or on a context for which no server location service is attached. The behavior changes depending on where the service is attached. Attaching the outbound service on a context without the server location service allows to bypass DNS resolving and use an outbound connection directly. Attaching the service after the server locator allows to use the DNS resolving before selecting an outbound connection.

The service can be configured with an ordered list of persistent connections to use. This list is used only when the outbound service is attached on a context without the server location. Such configuration allows to bypass DNS lookup and use an outbound connection directly. When outbound connections are configured, the content of the target (Request-URI or top Route) is ignored. The first available flow is selected and the peer address of the packet is set with the peer address of the selected flow. This means that sometimes a request can be sent on a flow that does not match the target of the request to send. This can happen if the outbound proxy does not record-route. For example let's say that the user-agent entity sends an INVITE request that is routed to P1 and then P2. The first index in the flow list would be P1. If P1 does not add itself as a Record-Route (but P2 does), the next requests sent on the dialog will be sent to P1 but with P2 being the target. When in this mode, the service also takes care of the failover. If there was a failure on the outbound flow selected to send the request, the next available flow will be selected. That failover is done by the service. The service simply blocks the request from being sent when there is no flow available (or when the entire flows in the list have been tried).

The outbound service fails to send a request when no flows list is set and no server location service is attached on the context.

Attaching the service AFTER the server locator allows load balancing as the DNS tells which target to try first when sending the request. In such mode, the failover is handled by the server locator. The service acts as the persistent connection service to the exception that it handles all parameters and headers related to outbound. When in this mode, setting a flow list has no effect as the list is not used at all

(DNS resolving tells which outbound connection to use).

If a DNS lookup is done and no persistent connection could be found in the flow list, then the stack could require to create a persistent connection. This is true only for non-REGISTER requests. For REGISTER requests, no persistent connection creation is requested by the stack. This connection creation process is necessary to be compliant to the draft-ietf-sip-outbound-15 section-4.3.

The persistent connection service MUST not be attached to a context on which the outbound connection service is also attached. It is also the responsibility of the application to not attach the server location after the outbound service. The application is responsible to add pre-loaded Route headers to the requests sent through persistent connections.

The **ISipOutboundConnectionSvc** is an ECOM object

The **ISipOutboundConnectionSvc** is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipOutboundConnectionSvc

Interface Id: IID_ISipOutboundConnectionSvc

A user can query the **ISipContext** (see page 23) to which this service is attached by calling **QueryIf** on it. It can also directly access all other services attached to the **ISipContext** (see page 23) through the same mean.

Location

SipCoreSvc/ISipOutboundConnectionSvc.h

See Also

ISipOutboundConnectionMgr (see page 100)

Methods

Method	Description
• A ConnectionCreationResult (see page 103)	Continues the processing to send the request over the newly created flow.
• A GetPersistentConnectionsPreferredOrder (see page 104)	Gets the list of persistent connections.
• A SetManager (see page 104)	Sets the manager.
• A SetPersistentConnectionsPreferredOrder (see page 104)	Sets the list of persistent connections to use.

Legend

•	Method
A	abstract

3.3.1.13.1 - Methods

3.3.1.13.1.1 - **ISipOutboundConnectionSvc::ConnectionCreationResult** Method

Continues the processing to send the request over the newly created flow.

C++

```
virtual mxt_result ConnectionCreationResult(IN mxt_opaque opqConnection, IN mxt_opaque opqId, IN mxt_result resCreation) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnection	The persistent connection identifier
IN mxt_opaque opqId	An opaque value that must be the same as the one given by the ISipOutboundConnectionMgr::EvConnectionNeeded (see page 100) method event.
IN mxt_result resCreation	resS_OK: The connection was created. The process of sending the request continues.
resFE_FAIL	The connection failed to be created or the application does not want to create one. The request will fail to be sent.

Returns

resS_OK: the processing continues. resFE_FAIL: there is no persistent connection matching this opaque or the service is not waiting for a connection creation

Description

This method must be called when the application returned eCREATE_CONNECTION_ASYNCHRONOUSLY upon the reporting of the event ISipOutboundConnectionMgr::EvConnectionNeeded (see page 100).

It must be called when the application has determined if the connection could be created or not. The connection has succeeded to be created when the event ISipPersistentConnectionList::EvEstablished is reported after that CSipPersistentConnectionList::Establish method has been called.

The connection has failed to be created when the event ISipPersistentConnectionListMgr::EvErrorOnConnection is called or when the application determined that no connection could be created for this remote destination.

See Also

ISipOutboundConnectionMgr::EvConnectionNeeded (see page 100)

3.3.13.1.2 - ISipOutboundConnectionSvc::GetPersistentConnectionsPreferredOrder Method

Gets the list of persistent connections.

C++

```
virtual const CVector<mxt_opaque>& GetPersistentConnectionsPreferredOrder() const = 0;
```

Returns

The persistent connections list.

Description

Gets the list of persistent connections.

3.3.13.1.3 - ISipOutboundConnectionSvc::SetManager Method

Sets the manager.

C++

```
virtual mxt_result SetManager(IN ISipOutboundConnectionMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipOutboundConnectionMgr* pMgr	The manager. It cannot be NULL.

Returns

resS_OK: manager is set. resFE_INVALID_PARAM: manager is NULL.

Description

Configures the manager of the outbound connection service.

3.3.13.1.4 - ISipOutboundConnectionSvc::SetPersistentConnectionsPreferredOrder Method

Sets the list of persistent connections to use.

C++

```
virtual mxt_result SetPersistentConnectionsPreferredOrder(IN CVector<mxt_opaque>& rvecConnections) = 0;
```

Parameters

Parameters	Description
IN CVector<mxt_opaque>& rvecConnections	The list of persistent connections identified through their opaque identifier.

Returns

resS_OK: list properly set. resFE_FAIL: the list could not be set.

Description

Sets the ordered list of persistent connections to use. This list is used only when the outbound connection service is attached on a context for which there is no server location service. If the outbound connection service is attached AFTER the server location service, then the list is not considered and the service does DNS resolving to find a persistent connection.

The first connection in the list is used to send REGISTER requests. All other connections are not considered to send REGISTER requests. The list is used to send requests other than REGISTERs and to do failover when some flows cannot be used because they are not currently connected or a failure has occurred when sending a request over a flow. Using a list allows to do redundancy.

The service only tries to use a connection configured in the list. When all connections in the list have been tried, the service will block packets from being sent through this service. The application will be able to send packets only when at least one connection of the list is re-connected again.

3.3.1.14 - ISipPersistentConnectionMgr Class

Class Hierarchy

ISipPersistentConnectionMgr

C++

```
class ISipPersistentConnectionMgr;
```

Description

The ISipPersistentConnectionMgr interface is the interface through which the CSipPersistentConnectionList reports events to the application about persistent connection. When the outbound connection service is attached, this interface is also used to receive the keep-alive service notifications.

Location

SipCoreSvc/ISipPersistentConnectionMgr.h

Methods

Method	Description
◆ A EvConnectionEstablished (see page 105)	Reports that a connection has been established.
◆ A EvConnectionTerminated (see page 106)	Reports the termination of a persistent connection.
◆ A EvErrorOnConnection (see page 106)	Reports a failure on a persistent connection.
◆ A EvSendResult (see page 106)	Reports the result of a sending operation on a flow.
◆ A EvStartKeepAliveResult (see page 107)	Informs of the StartKeepAlive result.

Legend

◆	Method
A	abstract

3.3.1.14.1 - Methods

3.3.1.14.1.1 - ISipPersistentConnectionMgr::EvConnectionEstablished Method

Reports that a connection has been established.

C++

```
virtual void EvConnectionEstablished(IN mxt_opaque opqConnection, IN const CSocketAddr& rLocalAddr, IN const CSocketAddr& rPeerAddr) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnection	The connection opaque received in the call to Establish.
IN const CSocketAddr& rLocalAddr	The effective local address used for the connection. If the local address was fully specified (address and port) in the call to Establish, rLocalAddr simply returns this same information. If however the port or address were left unspecified, rLocalAddr contains the local address and port that were actually used.

IN const CSocketAddr& rPeerAddr	The remote peer address. When a FQDN is used this address is assigned by the stack following a DNS query.
---------------------------------	---

Description

Reports the successful establishment of a persistent connection. This is called each time a persistent connection is either established for the first time or re-established after the connection went down.

See Also

[CSipPersistentConnectionList::Establish](#)

3.3.14.1.2 - **ISipPersistentConnectionMgr::EvConnectionTerminated** Method

Reports the termination of a persistent connection.

C++

```
virtual void EvConnectionTerminated(IN mxt_opaque opqConnection) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnection	The connection handle received in the call to Establish.

Description

Event reporting the successful termination of a persistent connection. This is reported after calling Terminate.

3.3.14.1.3 - **ISipPersistentConnectionMgr::EvErrorOnConnection** Method

Reports a failure on a persistent connection.

C++

```
virtual void EvErrorOnConnection(IN mxt_opaque opqConnection, IN mxt_result resConnectionError) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnection	The connection handle received in the call to Establish.
IN mxt_result resConnectionError	The error that occurred while trying to establish or re-establish the persistent connection.

Description

Reports that the stack was not able to establish or re-establish a persistent connection. This event is reported each time the stack tries to establish or re-establish a connection, thus it can be called multiple times for a single connection, until either the connection succeeds or Terminate is called for this connection.

3.3.14.1.4 - **ISipPersistentConnectionMgr::EvSendResult** Method New in 4.1.6

Reports the result of a sending operation on a flow.

C++

```
virtual void EvSendResult(IN mxt_opaque opqConnectionHandle, IN mxt_opaque opq, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnectionHandle	The persistent connection opaque identifier.
IN mxt_opaque opq	The opaque application data passed to the persistent connection list when calling Send.
IN mxt_result res	resS_OK: The sending was successful.
resFE_FAIL	The sending has failed. The application should consider this as a flow failure.

Description

Notifies the application about the sending result after that the application has requested some data to be sent on a flow.

The application should consider a flow to have failed when a failure result code is returned by this event. In such case, the application should call EvFlowFailure on the persistent connection list.

See Also

CSipPersistentConnectionList::Send CSipPersistentConnectionList::EvFlowFailure

3.3.14.1.5 - ISipPersistentConnectionMgr::EvStartKeepAliveResult Method New in 4.1.6

Informs of the StartKeepAlive result.

C++

```
virtual void EvStartKeepAliveResult(IN mxt_opaque opqConnectionHandle, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqConnectionHandle	The persistent connection opaque identifier.
IN mxt_result res	StartKeepAlive result. It can be resS_OK, if the keep alive was successfully started or resFE_FAIL if there was an error.

Description

Informs the manager about the StartKeepAlive() result.

This event is useful to determine if the keep alive mechanism has been successfully started for the specified connection or, if has failed, to know the error code.

See Also

CSipPersistentConnectionList::StartKeepAlive

3.3.15 - ISipPersistentConnectionSvc Class

Class Hierarchy



C++

```
class ISipPersistentConnectionSvc : public IEComUnknown;
```

Description

This service enables the support of persistent connections for the SIP context upon which it is attached. The requests sent by this service use a persistent connection only if the next hop URI (the top-most Route or the request-URI) corresponds to an established persistent connection.

Applications can create and manage their own connections with the CSipPersistentConnectionList.

When this service is attached to a SIP context, it overrides DNS SRV resolving and uses the resolved address used by the persistent connection instead.

This service must be attached to the SIP context after the ISipServerLocationSvc (see page 108), as it may have to override the next hop address in order to use the persistent connection.

The ISipPersistentConnectionSvc is an ECOM object

The ISipPersistentConnectionSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipPersistentConnectionSvc

Interface Id: IID_ISipPersistentConnectionSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipPersistentConnectionSvc.h

See Also

CSipPersistentConnectionList

3.3.1.16 - ISipServerLocationSvc Class

Class Hierarchy



C++

```
class ISipServerLocationSvc : public IEComUnknown;
```

Description

This service offers the functionality described in RFC 3263: Locating SIP Servers. This service is also used for transport selection, reconnection and failover to alternative transports. Hence, this service MUST always be attached to a context, unless the application knows it will use an established persistent connection.

It performs NAPTR, SRV, and A queries to resolve where SIP packets must be sent. It supports failover for requests when configured statefully.

DNS access is performed asynchronously by using a single thread for all queries. The core thread is not blocked while performing server location services, unless the stack is configured to have DNS access and the SIP core module share the same thread.

When used on a ISipContext (see page 23) with stateless services, this service must be configured as stateless through SetReqCtxServerLocationSvcMode (see page 109). Its default behaviour is to be stateful.

The ISipServerLocationSvc is an ECOM object

The ISipServerLocationSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipServerLocationSvc

Interface Id: IID_ISipServerLocationSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCore/ISipServerLocationSvc.h

Methods

Method	Description
• A SetReqCtxServerLocationSvcMode (see page 109)	Sets the server location service mode.
• A SetServerLocationListModifier (see page 109)	Sets a callback function for location list modifications.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
EServerLocationSvcMode (see page 109)	

3.3.1.16.1 - Methods

3.3.1.16.1.1 - **ISipServerLocationSvc::SetReqCtxServerLocationSvcMode** Method

Sets the server location service mode.

C++

```
virtual void SetReqCtxServerLocationSvcMode( IN EServerLocationSvcMode eMode ) = 0;
```

Description

Sets the server location service mode to stateful or stateless. When stateful, this service fails over when not able to send requests, while it does not do so when configured in stateless.

Warning

Not properly setting the server location service mode to eSTATELESS, when required, may result in misbehaviours. For example:

- Invalid branch value when forwarding requests/responses.

3.3.1.16.1.2 - **ISipServerLocationSvc::SetServerLocationListModifier** Method

Sets a callback function for location list modifications.

C++

```
virtual mxt_result SetServerLocationListModifier( IN mxt_PFNServerLocationListModifier pfnNew, IN mxt_opaque
opqCallbackParameter, OUT mxt_PFNServerLocationListModifier& rpfnPrevious, OUT mxt_opaque&
ropqPreviousParameter ) = 0;
```

Parameters

Parameters	Description
IN mxt_PFNServerLocationListModifier pfnNew	The new callback function for the location list modifications. It must not be NULL.
OUT mxt_PFNServerLocationListModifier& rpfnPrevious	The callback function that was already set in the service. This callback function should be called by the new callback method. It is not changed if this method fails. NULL if there was no callback function already set.
opqToPass	The opaque value passed to the callback function.
ropqToPassToPrevious	The opaque value that must be passed to rpfnPrevious when it is called. It is not changed if this method fails.
	This parameter must not be used if rpfnPrevious is NULL.

Returns

resFE_INVALID_ARGUMENT: pfnNew is NULL.

resS_OK: The callback method has been changed.

Description

Sets a new callback method for location list modifications. The callback function method is called with opqCallbackParameter after the asynchronous resolution has succeeded to allow another object to modify the record list.

The callback method returned by this method should be called with its opaque value in the new callback function.

The change in callback function only affects the new transactions. It is not used for current requests.

3.3.1.16.2 - **Enumerations**

3.3.1.16.2.1 - **ISipServerLocationSvc::EServerLocationSvcMode** Enumeration

```
enum EServerLocationSvcMode {
    eSTATEFUL,
    eSTATELESS
};
```

Description

Indicates how the server location service behaves when the requests fail to be sent.

Members

Members	Description
eSTATEFUL	Stateful mode: Server location service will fail-over when the stack is not able to send requests.
eSTATELESS	Stateless mode: Disable the fail-over mechanism.

3.3.1.17 - ISipStatelessDigestServerAuthSvc Class

Class Hierarchy



C++

```
class ISipStatelessDigestServerAuthSvc : public IEComUnknown;
```

Description

The digest server authentication service is a stateless service that allows to verify that incoming packets have been issued by a specific, authenticated user.

A manager of this service should provide incoming packets to this service in order to assert that the packet is correctly authenticated. This service reports an event to its manager to let it know how to further proceed.

It can be used to send both 401 and 407 responses.

This service supports the MD5 algorithm along with "auth" and "auth-int" quality of protection (qop).

In order to support the MD5-sess algorithm, the application must configure the authentication lifetime to zero with SetAuthenticationLifetime. It must also challenge incoming requests with the ChallengeRequest (see page 111) method that takes a packet, algorithm, and nonce as parameters.

The MD5-sess algorithm can be used with backend authentication services like diameter or radius.

The ISipStatelessDigestServerAuthSvc is an ECOM object

The ISipStatelessDigestServerAuthSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipStatelessDigestServerAuthSvc

Interface Id: IID_ISipStatelessDigestServerAuthSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipStatelessDigestServerAuthSvc.h

Methods

Method	Description
• A ChallengeRequest (see page 111)	Challenges the specified request.
• A ContainsCredentials (see page 112)	Verifies if the SIP packet contains credentials, and the validity of some of them.
• A GetMd5AlgoHash (see page 112)	Retrieves the MD5 hash of A1 when the MD5 algorithm is used.
• A QopMustBePresent (see page 113)	Configures whether or not the presence of the qop parameter in the authorization header must be checked and its value validated, for backward compatibility purposes.
• A RejectRequest (see page 113)	Rejects the specified request.
• A SetAuthLifetime (see page 114)	Configures the time before authentication information becomes stale.
• A SetPrivateKey (see page 114)	Configures the private key used in nonce encoding.
• A SetRequestedQualityOfProtection (see page 114)	Configures the quality of protection sent in challenges.
• A SetSupportedRealm (see page 115)	Configures the realm supported by this device.
• A VerifyAuthentication (see page 115)	Verifies the credentials found in the SIP packet.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
EAlgorithm (see page 116)	
EQualityOfProtection (see page 116)	

3.3.1.17.1 - Methods

3.3.1.17.1.1 - **ISipStatelessDigestServerAuthSvc::ChallengeRequest** Method

Challenges the specified request.

C++

```
virtual mxt_result ChallengeRequest(IN const CSipPacket& rRequest, IN unsigned int uChallengeType, IN EAlgorithm eAlgorithm, IN const CString& rstrNonce, IN bool bIsStale) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request to challenge.
IN unsigned int uChallengeType	The challenge type (the response's status code: uUNAUTHORIZED (see page 393) or uPROXY_AUTHENTICATION_REQUIRED (see page 393)).
IN EAlgorithm eAlgorithm	The algorithm to use for the challenge.
IN const CString& rstrNonce	The new nonce to use for the challenge.
IN bool bIsStale	Whether or not there was a stale nonce in the request.

Returns

resFE_INVALID_ARGUMENT: The challenge type is wrong or the packet is not a valid request.

resFE_INVALID_STATE: The parent context is not set.

resFE_FAIL: The challenge could not be sent by the transport layer for some reason.

resS_OK: The challenge has been successfully sent.

Description

Challenges the specified request with the specified algorithm and nonce.

This method MUST be used when SetAuthenticationLifetime has been configured with a value of zero. This means the user of the service (or some other mechanism) is responsible for generating the nonce.

3.3.1.17.1.1.2 - **ISipStatelessDigestServerAuthSvc::ChallengeRequest** Method

Challenges the specified request.

C++

```
virtual mxt_result ChallengeRequest(IN const CSipPacket& rRequest, IN unsigned int uChallengeType, IN bool bIsStale) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request to challenge.
IN unsigned int uChallengeType	The challenge type (the response's status code: uUNAUTHORIZED (see page 393) or uPROXY_AUTHENTICATION_REQUIRED (see page 393)).

IN bool bIsStale	Whether or not there was a stale nonce in the request.
------------------	--

Returns

resFE_INVALID_ARGUMENT: The packet is not a valid request.
 resFE_INVALID_STATE: The parent context is not set.
 resFE_FAIL: The challenge could not be sent by the transport layer for some reason.
 resS_OK: The challenge has been successfully sent.

Description

Issues a challenge to the specified request.

This method can only be used to issue challenges with the MD5 algorithm. The other version of ChallengeRequest must be used to challenge requests with the MD5-sess algorithm.

ACK and CANCEL requests must never be challenged.

3.3.1.17.1.2 - ISipStatelessDigestServerAuthSvc::ContainsCredentials Method

Verifies if the SIP packet contains credentials, and the validity of some of them.

C++

```
virtual mxt_result ContainsCredentials(IN const CSipPacket& rRequest, OUT CString& rstrUsername) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request that contains the credentials for the realm managed by this object.
OUT CString& rstrUsername	The username included in the request for the realm managed by this object.

Returns

resS_OK: The packet contains credentials.
 resFE_FAIL: The packet does not contain credentials, or no (Proxy-)Authorization header with the right realm has been found.
 resFE_SIPCORESVC_STALE_NONCE: The packet contains credentials but the nonce is stale.
 resFE_INVALID_ARGUMENT: One of the packet's parameters is wrong or missing, or the packet contains some error.

Description

Checks if the SIP packet contains credentials, and if so, also checks whether or not the nonce is stale and has been generated locally. Also checks the presence of the username, cnonce, and nonce-count (nc) parameters when required; the presence and validity of the realm, algorithm, and qop (quality of protection) parameters; and the presence of the nonce plus its not being stale.

See Also

VerifyAuthentication (see page 115)

3.3.1.17.1.3 - ISipStatelessDigestServerAuthSvc::GetMd5AlgoHash Method

Retrieves the MD5 hash of A1 when the MD5 algorithm is used.

C++

```
virtual mxt_result GetMd5AlgoHash(IN const CString& rstrUsername, IN const CString& rstrPassword, OUT CString& rstrHashA1) const = 0;
```

Parameters

Parameters	Description
IN const CString& rstrUsername	The username.
IN const CString& rstrPassword	The password.
rstrHash	String that contains the hash upon return.

Returns

See `MxCalculateMd5Checksum` in `SipParser/Authentication.cpp`

Description

Generates a hash of A1 according to RFC 2617. The realm used is the realm configured through `SetSupportedRealm` (see page 115).

See Also

`SipParser/Authentication.cpp`: `MxCalculateMd5Checksum`

3.3.1.17.1.4 - ISipStatelessDigestServerAuthSvc::QopMustBePresent Method

Configures whether or not the presence of the qop parameter in the authorization header must be checked and its value validated, for backward compatibility purposes.

C++

```
virtual void QopMustBePresent(IN bool bPresent) = 0;
```

Parameters

Parameters	Description
IN bool bPresent	True if the qop parameter must be present, false otherwise.

Description

Configures whether or not the presence of the qop parameter in the authorization header must be checked and its value validated, for backward compatibility purposes.

In the affirmative, if the proper quality of protection is not met, the request is rejected with a "403 Forbidden" response.

Implementors must know that calling `QopMustBePresent` with true breaks the backward compatibility with RFC 2069.

See Also

`SetRequestedQualityOfProtection` (see page 114)

3.3.1.17.1.5 - ISipStatelessDigestServerAuthSvc::RejectRequest Method

Rejects the specified request.

C++

```
virtual mxt_result RejectRequest(IN const CSipPacket& rRequest, IN unsigned int uResponseType, IN const char* pszReasonPhrase) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request to challenge.
IN unsigned int uResponseType	The response type; the response's status code MUST be uFORBIDDEN (see page 393) otherwise the method will fail with <code>resFE_INVALID_ARGUMENT</code> .
IN const char* pszReasonPhrase	Reason phrase to use in the sent response. If NULL, the default reason phrase for the specified <code>uStatusCode</code> is used.

Returns

`resFE_INVALID_ARGUMENT`: Response type is wrong or the packet is not a valid request.

`resFE_INVALID_STATE`: The parent context is not set.

`resFE_FAIL`: The response could not be sent by the transport layer for some reason.

`resS_OK`: The response was successfully sent.

Description

This method will have the effect of sending a final negative response to the received request. The received packet is then dropped and no further processing can be made on it.

3.3.1.17.1.6 - ISipStatelessDigestServerAuthSvc::SetAuthLifetime Method

Configures the time before authentication information becomes stale.

C++

```
virtual void SetAuthLifetime(IN unsigned int uExpirationMin) = 0;
```

Parameters

Parameters	Description
IN unsigned int uExpirationMin	The time, in minutes, that the stack should consider a nonce valid. If zero, the stack considers all nonces it receives as valid and never re-challenges a request because of stale authentication information.

Description

This method allows to configure the length of time that authentication information provided by the stack is to be considered valid. This serves as a mean to periodically re-challenge incoming requests, which makes some attacks somewhat harder.

If uExpirationMin is configured to zero, this service does not manage stale authentication information. The application may do so if it wants.

This configures the time that a generated nonce is to remain valid. When this service receives a request that uses a nonce that has been generated by the stack but is too old, then a new challenge is automatically issued.

The default value for this is 5 minutes.

3.3.1.17.1.7 - ISipStatelessDigestServerAuthSvc::SetPrivateKey Method

Configures the private key used in nonce encoding.

C++

```
virtual void SetPrivateKey(IN const CString& rstrPrivateKey) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrPrivateKey	The private key used in nonce encoding.

Description

Configures the private key used in nonce encoding.

3.3.1.17.1.8 - ISipStatelessDigestServerAuthSvc::SetRequestedQualityOfProtection Method

Configures the quality of protection sent in challenges.

C++

```
virtual mxt_result SetRequestedQualityOfProtection(IN unsigned int bsQopOptions) = 0;
```

Parameters

Parameters	Description
IN unsigned int bsQopOptions	A bitset combination of the EQualityOfProtection (see page 116) values. eNone can only be used alone.

Returns

resS_OK: When the bitset contains accepted values.

resFE_INVALID_ARGUMENT: Otherwise.

Description

Configures the quality of protection requested in challenges. This is an optional parameter in RFC 2617 to remain backward compatible with RFC 2069.

When eNone is used, no qop option is sent in the challenge.

See Also

[QopMustBePresent](#) (see page 113)

3.3.1.17.1.9 - **ISipStatelessDigestServerAuthSvc::SetSupportedRealm** Method

Configures the realm supported by this device.

C++

```
virtual void SetSupportedRealm(IN const CString& rstrRealm) = 0;
```

Parameters

Parameters	Description
strRealm	The realm supported by this service.

Description

This method configures the realm for which this service manages authentication. Incoming requests are searched for authentication information for this specific realm.

No default configuration exists for this setting, it MUST be called with the proper realm for this service to work correctly.

3.3.1.17.1.10 - **ISipStatelessDigestServerAuthSvc::VerifyAuthentication** Method

Verifies the credentials found in the SIP packet.

C++

```
virtual mxt_result VerifyAuthentication(IN const CSipPacket& rRequest, IN const CString& rstrHashA1) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request that contains the credentials for this realm.
IN const CString& rstrHashA1	Hash of A1, as specified in RFC 2617. See below for more information.

Returns

resFE_ABORT: A problem has been encountered when computing the checksum.

resFE_INVALID_ARGUMENT: One of the packet's other parameters is wrong or missing or the packet contains some error.

resFE_FAIL: The response is invalid.

resS_OK: Authentication has succeeded.

Description

Verifies the credentials found in the packet.

The string rstrHashA1 must be built as follows (refer to RFC 2617 for more details).

```
MD5HASH(username ":" realm ":" password)
```

An hash of A1 can easily be obtained if the password is available in clear text through the [GetMd5AlgoHash](#) (see page 112).

Currently, only the verification of the algorithm MD5 is supported.

Applications should consider storing hashes of user password instead of directly storing the passwords in clear text. Storing a hash of A1 instead of storing the password is usually the proper thing to do when not using some form of authentication back-end.

See Also

[GetMd5AlgoHash](#) (see page 112), [RemoveVerifiedHeader](#)

3.3.1.17.2 - **Enumerations**

3.3.1.17.2.1 - **ISipStatelessDigestServerAuthSvc::EAlgorithm** Enumeration

```
enum EAlgorithm {
    eMD5,
    eMD5_Session
};
```

Description

Indicates the algorithm to use for the challenge.

Members

Members	Description
eMD5	Use the MD5 algorithm.
eMD5_Session	Use the MD5-session algorithm.

3.3.1.17.2.2 - **ISipStatelessDigestServerAuthSvc::EQualityOfProtection** Enumeration

```
enum EQualityOfProtection {
    eNone = 0x00000000,
    eAuth = 0x00000001,
    eAuthInt = 0x00000002,
    eQopMask = 0x00000003
};
```

Description

Indicates the quality of protection requested in challenges. This is an optional parameter in RFC 2617 to remain backward compatible with RFC 2069.

Members

Members	Description
eNone = 0x00000000	Don't use authentication.
eAuth = 0x00000001	Use the authentication scheme.
eAuthInt = 0x00000002	Use the authentication with integrity protection.
eQopMask = 0x00000003	The combination of all possible qops, for validation purposes.

3.3.1.18 - **ISipStatisticsInfo** Class

Class Hierarchy



C++

```
class ISipStatisticsInfo : public IEComUnknown;
```

Description

This interface is used to get statistics about sent and received packets, ongoing transactions, and DNS queries.

The **ISipStatisticsInfo** is an ECOM object

The **ISipStatisticsInfo** is an ECOM object that is accessed through the following ECOM identifier:

Interface Id: **IID_ISipStatisticsInfo**

A user can call **QueryIf** on a **CLSID_CSipStatisticsContainer** object created with the **CreateEComInstance** method to get the **ISipStatisticsInfo** interface of the object.

Location

[SipCoreSvc/ISipStatisticsInfo.h](#)

Methods

Method	Description
• A GetNumActiveTransactions (see page 117)	Gets the total number of active transactions.
• A GetNumDnsQueriesFailed (see page 117)	Gets the number of times a DNS query has failed.
• A GetNumDnsQueriesSucceeded (see page 118)	Gets the number of times a DNS query has succeeded.
• A GetNumFinalResponseRetransmissionsRx (see page 118)	Gets the number of times a final response to the specified SIP method has been received and processed as a retransmission.
• A GetNumFinalResponseRetransmissionsTx (see page 118)	Gets the number of times a final response to the specified SIP method has been retransmitted.
• A GetNumFinalResponseRx (see page 118)	Gets the number of times the specified final response has been received.
• A GetNumFinalResponseTx (see page 119)	Gets the number of times the specified final response has been sent.
• A GetNumNonFinalResponseRetransmissionsTx (see page 119)	Gets the number of times a non-final response to the specified SIP method has been retransmitted.
• A GetNumNonFinalResponseRx (see page 119)	Gets the number of times a non-final response to the specified SIP method has been received.
• A GetNumNonFinalResponseTx (see page 120)	Gets the number of times the specified non-final response has been sent.
• A GetNumRequestRetransmissionsRx (see page 120)	Gets the number of times the specified SIP method has been received and processed as a retransmission.
• A GetNumRequestRetransmissionsTx (see page 120)	Gets the number of times the specified SIP method has been retransmitted.
• A GetNumRequestsRx (see page 121)	Gets the number of times the specified SIP method has been received.
• A GetNumRequestsTx (see page 121)	Gets the number of times the specified SIP method has been sent.
• A GetTotalNumRequestsRx (see page 121)	Gets the total number of requests received.
• A GetTotalNumRequestsTx (see page 121)	Gets the total number of requests sent.
• A GetTotalNumResponsesRx (see page 122)	Gets the total number of responses received.
• A GetTotalNumResponsesTx (see page 122)	Gets the total number of responses sent.
• A GetTotalNumTransactions (see page 122)	Gets the total number of transactions.
• A Reset (see page 122)	Resets all statistics.

Legend

•	Method
A	abstract

3.3.1.18.1 - Methods

3.3.1.18.1.1 - ISipStatisticsInfo::GetNumActiveTransactions Method

Gets the total number of active transactions.

C++

```
virtual unsigned int GetNumActiveTransactions() const = 0;
```

Returns

The total number of active transactions.

Description

Gets the total number of active transactions. An ACK request to a 2xx INVITE response is considered as a different transaction than the INVITE, while an ACK to a non 2xx is considered as the same transaction as the INVITE transaction.

3.3.1.18.1.2 - ISipStatisticsInfo::GetNumDnsQueriesFailed Method

Gets the number of times a DNS query has failed.

C++

```
virtual unsigned int GetNumDnsQueriesFailed() const = 0;
```

Returns

The number of times a DNS query has failed.

Description

Gets the number of times a DNS query has failed. DNS queries done by the CSipPersistentConnectionList, CSipPrivacySvc, and CSipUaAssertedIdentitySvc are not included.

3.3.1.18.1.3 - ISipStatisticsInfo::GetNumDnsQueriesSucceeded Method

Gets the number of times a DNS query has succeeded.

C++

```
virtual unsigned int GetNumDnsQueriesSucceeded() const = 0;
```

Returns

The number of times a DNS query has succeeded.

Description

Gets the number of times a DNS query has succeeded. DNS queries done by the CSipPersistentConnectionList, CSipPrivacySvc, and CSipUaAssertedIdentitySvc are not included.

3.3.1.18.1.4 - ISipStatisticsInfo::GetNumFinalResponseRetransmissionsRx Method

Gets the number of times a final response to the specified SIP method has been received and processed as a retransmission.

C++

```
virtual unsigned int GetNumFinalResponseRetransmissionsRx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a final response to the specified SIP method has been received and processed as a retransmission.

Description

Gets the number of times a final response to the specified SIP method has been received and processed as a retransmission.

3.3.1.18.1.5 - ISipStatisticsInfo::GetNumFinalResponseRetransmissionsTx Method

Gets the number of times a final response to the specified SIP method has been retransmitted.

C++

```
virtual unsigned int GetNumFinalResponseRetransmissionsTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a final response to the specified SIP method has been retransmitted.

Description

Gets the number of times a final response to the specified SIP method has been retransmitted.

3.3.1.18.1.6 - ISipStatisticsInfo::GetNumFinalResponseRx Method

Gets the number of times the specified final response has been received.

C++

```
virtual unsigned int GetNumFinalResponseRx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a final response to the specified SIP method has been received.

Description

Gets the number of times the specified final response has been received. This excludes retransmissions.

3.3.1.18.1.7 - ISipStatisticsInfo::GetNumFinalResponseTx Method

Gets the number of times the specified final response has been sent.

C++

```
virtual unsigned int GetNumFinalResponseTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a final response to the specified SIP method has been sent.

Description

Gets the number of times the specified final response has been sent. This excludes retransmissions.

3.3.1.18.1.8 - ISipStatisticsInfo::GetNumNonFinalResponseRetransmissionsTx Method

Gets the number of times a non-final response to the specified SIP method has been retransmitted.

C++

```
virtual unsigned int GetNumNonFinalResponseRetransmissionsTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a non-final response to the specified SIP method has been retransmitted.

Description

Gets the number of times a non-final response to the specified SIP method has been retransmitted. This occurs if a non-final response is sent to a request and a retransmission of the request is received. The stack sends back the non-final response again, thus it is counted as a retransmission.

3.3.1.18.1.9 - ISipStatisticsInfo::GetNumNonFinalResponseRx Method

Gets the number of times a non-final response to the specified SIP method has been received.

C++

```
virtual unsigned int GetNumNonFinalResponseRx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a non-final response to the specified SIP method has been received.

Description

Gets the number of times a non-final response to the specified SIP method has been received.

3.3.1.18.1.10 - ISipStatisticsInfo::GetNumNonFinalResponseTx Method

Gets the number of times the specified non-final response has been sent.

C++

```
virtual unsigned int GetNumNonFinalResponseTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times a non-final response to the specified SIP method has been sent.

Description

Gets the number of times the specified non-final response has been sent. This excludes retransmissions.

3.3.1.18.1.11 - ISipStatisticsInfo::GetNumRequestRetransmissionsRx Method

Gets the number of times the specified SIP method has been received and processed as a retransmission.

C++

```
virtual unsigned int GetNumRequestRetransmissionsRx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times the specified SIP method has been received and processed as a retransmission.

Description

Gets the number of times the specified SIP method has been received and processed as a retransmission.

3.3.1.18.1.12 - ISipStatisticsInfo::GetNumRequestRetransmissionsTx Method

Gets the number of times the specified SIP method has been retransmitted.

C++

```
virtual unsigned int GetNumRequestRetransmissionsTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times the specified SIP method has been retransmitted.

Description

Gets the number of times the specified SIP method has been retransmitted.

3.3.1.18.1.13 - ISipStatisticsInfo::GetNumRequestsRx Method

Gets the number of times the specified SIP method has been received.

C++

```
virtual unsigned int GetNumRequestsRx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times the specified SIP method has been received.

Description

Gets the number of times the specified SIP method has been received. This excludes retransmissions.

3.3.1.18.1.14 - ISipStatisticsInfo::GetNumRequestsTx Method

Gets the number of times the specified SIP method has been sent.

C++

```
virtual unsigned int GetNumRequestsTx(IN const ESipMethod eMethod) const = 0;
```

Parameters

Parameters	Description
IN const ESipMethod eMethod	The SIP method.

Returns

The number of times the specified SIP method has been sent.

Description

Gets the number of times the specified SIP method has been sent. This excludes retransmissions.

3.3.1.18.1.15 - ISipStatisticsInfo::GetTotalNumRequestsRx Method

Gets the total number of requests received.

C++

```
virtual unsigned int GetTotalNumRequestsRx() const = 0;
```

Returns

The total number of requests received.

Description

Gets the total number of requests received, including retransmissions.

3.3.1.18.1.16 - ISipStatisticsInfo::GetTotalNumRequestsTx Method

Gets the total number of requests sent.

C++

```
virtual unsigned int GetTotalNumRequestsTx() const = 0;
```

Returns

The total number of requests sent.

Description

Gets the total number of requests sent, including retransmissions.

3.3.1.18.1.17 - **ISipStatisticsInfo::GetTotalNumResponsesRx** Method

Gets the total number of responses received.

C++

```
virtual unsigned int GetTotalNumResponsesRx() const = 0;
```

Returns

The total number of responses received.

Description

Gets the total number of responses received, including retransmissions.

3.3.1.18.1.18 - **ISipStatisticsInfo::GetTotalNumResponsesTx** Method

Gets the total number of responses sent.

C++

```
virtual unsigned int GetTotalNumResponsesTx() const = 0;
```

Returns

The total number of responses sent.

Description

Gets the total number of responses sent, including retransmissions.

3.3.1.18.1.19 - **ISipStatisticsInfo::GetTotalNumTransactions** Method

Gets the total number of transactions.

C++

```
virtual unsigned int GetTotalNumTransactions() const = 0;
```

Returns

The total number of transactions.

Description

Gets the total number of transactions. This includes terminated and active transactions.

3.3.1.18.1.20 - **ISipStatisticsInfo::Reset** Method

Resets all statistics.

C++

```
virtual void Reset() = 0;
```

Description

Resets all statistics to zero.

3.3.1.19 - ISipStatisticsSvc Class

Class Hierarchy



C++

```
class ISipStatisticsSvc : public IEComUnknown;
```

Description

This service is responsible to gather statistical data of packets that are sent and received on the associated context.

Stateless vs stateful In stateful mode, for an incoming INVITE or an outgoing ACK, a new statistics request context service is created and associated with the request context. It can detect retransmitted 2xx to INVITE, retransmitted ACK request, and retransmitted reliable 1xx. Set to stateful for a transaction stateful proxy or user agent.

Less memory is used in stateless mode. Set to stateless for a stateless proxy context.

If a CSipStatisticsContainer (see page 78) is set on the service with SetTransactionStatistics (see page 124), statistics are accumulated in the associated container.

This service must be attached to the context after the CSipSessionSvc and CSipServerLocationSvc to be able to gather statistics properly.

The ISipStatisticsSvc is an ECOM object

The ISipStatisticsSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipStatisticsSvc

Interface Id: IID_ISipStatisticsSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipStatisticsSvc.h

Methods

Method	Description
• A SetServiceMode (see page 123)	Configures the service for stateful or stateless operation.
• A SetTransactionStatistics (see page 124)	Configures the transaction statistics to be used.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
EServiceMode (see page 124)	

3.3.1.19.1 - Methods

3.3.1.19.1.1 - ISipStatisticsSvc::SetServiceMode Method

Configures the service for stateful or stateless operation.

C++

```
virtual void SetServiceMode(IN const EServiceMode eMode) = 0;
```

Parameters

Parameters	Description
IN const EServiceMode eMode	Either of eSTATEFUL or eSTATELESS.

Description

Configures how the service behaves under certain conditions. The service uses less resources when behaving statelessly, but some retransmissions will not be detected. Set the mode to eSTATEFUL for transaction stateful proxy and user agents. Set to eSTATELESS for a stateless proxy.

3.3.1.19.1.2 - ISipStatisticsSvc::SetTransactionStatistics Method

Configures the transaction statistics to be used.

C++

```
virtual void SetTransactionStatistics(IN ISipTransactionStatistics* pTransStats) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatistics* pTransStats	Pointer to the transaction statistics.

Description

Configures the transaction statistics to use with this context. If the statistic service is configured after the server location service is attached to a context, then the statistics service automatically sets the proper transaction statistics in the server location service.

3.3.1.19.2 - Enumerations**3.3.1.19.2.1 - ISipStatisticsSvc::EServiceMode Enumeration**

```
enum EServiceMode {
    eSTATEFUL,
    eSTATELESS
};
```

Description

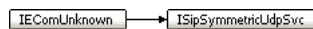
Used to configure the service for stateful or stateless operation.

See Also

SetServiceMode (see page 123)

Members

Members	Description
eSTATEFUL	Sets the mode to eSTATEFUL for transaction stateful proxy and user agents.
eSTATELESS	Set to eSTATELESS for a stateless proxy.

3.3.1.20 - ISipSymmetricUdpSvc Class**Class Hierarchy****C++**

```
class ISipSymmetricUdpSvc : public IEComUnknown;
```

Description

This service is used to override the network-level local port used to send packets over UDP. When attached to a context, the service will use the first UDP listening port of the network interface selected to send the packet.

This service is mostly used when Proxies and Registrars expect SIP traffic to come from a certain port, the most common example being the same as the listening port, e.g. 5060.

Location

SipCoreSvc/ISipSymmetricUdpSvc.h

See Also

SipCoreSvc/CSymmetricUdpSvc.h

3.3.1.21 - ISipViaManagementSvc Class New in 4.1.6

This service is responsible to set the top-most Via header.

Class Hierarchy



C++

```
class ISipViaManagementSvc : public IEComUnknown;
```

Description

This service sets the top-most Via header in the packet, if it is not already present.

Notes

This service is mandatory. It must be the last connection service attached to a context. The only exception to this is if the output controlling service is attached it must be attached after this.

The ISipViaManagementSvc is an ECOM object

The ISipViaManagementSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipViaManagementSvc

Interface Id: IID_ISipViaManagementSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipCoreSvc/ISipViaManagementSvc.h

Methods

Method	Description
SetViaManagementSvcMode (see page 126)	Sets the Via management service mode.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
EViaManagementMode (see page 126)	Possible via management modes.

3.3.1.21.1 - Methods

3.3.1.21.1.1 - ISipViaManagementSvc::SetViaManagementSvcMode Method

Sets the Via management service mode.

C++

```
virtual void SetViaManagementSvcMode(EViaManagementMode eMode) = 0;
```

Parameters

Parameters	Description
EViaManagementMode eMode	The updated mode.

Description

Sets the Via management service mode. When the service is stateless, the branch parameter remains unchanged when this service is used by a proxy implementation.

3.3.1.21.2 - Enumerations

3.3.1.21.2.1 - ISipViaManagementSvc::EViaManagementMode Enumeration

Possible via management modes.

C++

```
enum EViaManagementMode {
    eSTATEFUL,
    eSTATELESS
};
```

Description

These are the possible via management modes that can be set using ISipViaManagementSvc::SetViaManagementSvcMode (see page 126).

Members

Members	Description
eSTATEFUL	Stateful mode: Via management service will fail-over when the stack is not able to send requests. This is the default mode.
eSTATELESS	Stateless mode: Disable the fail-over mechanism.

3.3.2 - Enumerations

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

3.3.3 - Types

This section documents the types of the Sources/SipCoreSvc folder.

Types

Type	Description
mxt_PFNServerLocationListModifier (see page 126)	Callback to a list modifier.

3.3.3.1 - mxt_PFNServerLocationListModifier Type

Callback to a list modifier.

C++

```
typedef void (* mxt_PFNServerLocationListModifier)(IN const CHostPort& rFqdn, INOUT CList<SNaptrRecord>& rlstNaptrRecord, IN mxt_opaque opq);
```

Parameters

Parameters	Description
rFqdn	The address being resolved.
rlstNaptrRecord	The list of NAPTR records.
opq	An opaque.

Description

This data type defines a function prototype for a list modifier, which can modify the list, removing or adding records, or simply modifying the order of the records created by the resolution service. The only restriction is that the modification be made without changing the execution context.

Location

SipCoreSvc/CSipReqCtxServerLocationSvc.h

3.4 - SipParser

This section documents the Sources/SipParser folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 127)
- Enumerations (see page 384)
- Functions (see page 388)
- Variables (see page 391)

3.4.1 - Classes

This section documents the classes of the Sources/SipParser folder.

Classes

Class	Description
CAbsoluteUri (see page 128)	
CDate (see page 133)	
CGenericParam (see page 140)	Class: CGenericParam
CGenParamList (see page 146)	
CHostList (see page 155)	
CHostPort (see page 167)	
ClmUri (see page 177)	
CMailboxUri (see page 182)	
CMessageSummary (see page 192)	
CNameAddr (see page 201)	
CPresUri (see page 211)	
CQuotedString (see page 215)	
CRawHeader (see page 218)	
CReginfo (see page 222)	
CRequestLine (see page 234)	
CSipHeader (see page 239)	Class: CSipHeader
CSipMessageBody (see page 298)	
CSipPacketParser (see page 312)	
CSipStatusLine (see page 321)	
CSipUri (see page 325)	
CStringHelper (see page 341)	
CTelUri (see page 351)	
CToken (see page 360)	
CUriFactory (see page 378)	
IUri (see page 380)	

3.4.1.1 - CAbsoluteUri Class

Class Hierarchy



C++

```
class CAbsoluteUri : public IUri;
```

Description

This class abstracts a basic URI as per RFC 3261 ABNF (derived from RFC 2396)

```

RFC 3261 ABNF
absoluteURI  = scheme ":" ( hier-part / opaque-part )
hier-part    = ( net-path / abs-path ) [ "?" query ]
net-path     = "://" authority [ abs-path ]
abs-path     = "/" path-segments
opaque-part  = uric-no-slash *uric
uric         = reserved / unreserved / escaped
uric-no-slash = unreserved / escaped / ";" / "?" / ":" / "@"
                / "&" / "=" / "+" / "$" / ","
path-segments = segment *( "/" segment )
segment      = *pchar *( ";" param )
param        = *pchar
pchar         = unreserved / escaped /
                ":" / "@" / "&" / "=" / "+" / "$" / ","
scheme       = ALPHA *( ALPHA / DIGIT / "+" / "-" / ".")
authority    = srvr / reg-name
srvr         = [ userinfo "@" ] hostport ]
reg-name     = 1*( unreserved / escaped / "$" / ","
                / ";" / ":" / "@" / "&" / "=" / "+")
query        = *uric
  
```

The CAbsoluteUri is used as a last resort when parsing URIs. When all URIs that are natively supported by the SIP stack have been tried, the unknown URI is parsed as a CAbsoluteUri.

No processing is made on the CAbsoluteUri when it is parsed; it is kept in escaped form, as was received.

Note that there is no specific parsing of the (hier-part / opaque part) ABNF. Anything that is found after the colon trailing the scheme is considered as the CAbsoluteUri's body.

To parse CAbsoluteUri, users should use the CUriFactory (see page 378) to generate and parse.

Location

SipParser/CAbsoluteUri.h

See Also

IUri (see page 380), CSipUri (see page 325), CUriFactory (see page 378)

Constructors

Constructor	Description
CAbsoluteUri (see page 129)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CAbsoluteUri (see page 130)	Destructor.

IUri Class

IUri Class	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 133)	Assignment operator.

Legend

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

Methods

Method	Description
  GenerateCopy (see page 130)	Generates a copy of this URI.
  GetBody (see page 130)	Provides access to the body of the URI, in the form in which it was received from the network.
  GetScheme (see page 131)	Returns the URI's scheme. In this case, the m_strScheme data member.
  GetUriType (see page 131)	Returns URI type, SIP or SIPS.
  IsEquivalent (see page 131)	Compares the given URI with this instance using applicable RFC rules. For this URI type, the scheme is compared case-insensitively and the body byte-per-byte.
  Parse (see page 132)	Parses a byte string into usable data.
  Reset (see page 132)	Reinitializes the instance.
  Serialize (see page 132)	Outputs the data member in a format that is ready to be sent on the network.
  SetScheme (see page 133)	Sets the scheme part of the URI.

IUri Class

IUri Class	Description
  GenerateCopy (see page 381)	Generates a copy of this URI.
  GetScheme (see page 382)	Returns the URI's scheme.
  GetUriType (see page 382)	Returns the URI type.
  IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
  Parse (see page 382)	Parses a byte string into usable data.
  Reset (see page 383)	Reinitializes the instance.
  Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	virtual
	abstract

Enumerations

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.1.1 - Constructors

3.4.1.1.1.1 - CAbsoluteUri

3.4.1.1.1.1.1 - CAbsoluteUri::CAbsoluteUri Constructor

Constructor.

C++

```
CAbsoluteUri();
```

Description

Default constructor.

3.4.1.1.1.2 - CAbsoluteUri::CAbsoluteUri Constructor

Copy Constructor.

C++

```
CAbsoluteUri(IN const CAbsoluteUri& rSrc);
```

Parameters

Parameters	Description
IN const CAbsoluteUri& rSrc	URI to copy.

Description

Copy constructor.

3.4.1.1.2 - Destructors**3.4.1.1.2.1 - CAbsoluteUri::~CAbsoluteUri Destructor**

Destructor.

C++

```
virtual ~CAbsoluteUri();
```

Description

Destructor.

3.4.1.1.3 - Methods**3.4.1.1.3.1 - CAbsoluteUri::GenerateCopy Method**

Generates a copy of this URI.

C++

```
virtual GO IUuri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.1.3.2 - GetBody**3.4.1.1.3.2.1 - CAbsoluteUri::GetBody Method**

Provides access to the body of the URI, in the form in which it was received from the network.

C++

```
const CString& GetBody() const;
```

```
CString& GetBody();
```

Returns

Body of the URI.

Description

Provides access to the as-received-from network version of the the URI's body.

3.4.1.1.3.3 - CAbsoluteUri::GetScheme Method

Returns the URI's scheme. In this case, the m_strScheme data member.

C++

```
virtual const char* GetScheme() const;
```

Returns

Returns the m_strScheme member.

Description

Provides access to the m_strScheme member.

See Also

[SetScheme](#) (see page 133)

3.4.1.1.3.4 - CAbsoluteUri::GetUriType Method

Returns URI type, SIP or SIPS.

C++

```
virtual EUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the [IsEquivalent](#) (see page 131) method.

3.4.1.1.3.5 - CAbsoluteUri::IsEquivalent Method

Compares the given URI with this instance using applicable RFC rules. For this URI type, the scheme is compared case-insensitively and the body byte-per-byte.

C++

```
virtual bool IsEquivalent(IN const IUri& rSrc) const;
```

Parameters

Parameters	Description
IN const IUri& rSrc	URI against which to compare.

Returns

True if the given URI is equivalent to this instance.

Description

Compares the URIs. The scheme is compared case-insensitively, and the body is compared byte-per-byte. Note that this method must only be called for CAbsoluteUri (see page 128) instances.

3.4.1.1.3.6 - CAbsoluteUri::Parse Method

Parses a byte string into usable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	A value of IUri::eALLOW_SPECIAL_CHARS indicates that the URI can consider the comma ',', question mark '?', or semi-colon ';' characters as part of this URI. This is as per RFC 3261 conformance item {811} (URLs that contain one of these characters MUST be within angle quotes). This restriction extends to all (name-addr addr-spec) constructs (including SIP extensions).
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. Initially, rpcPos points after the colon following the scheme. If Parse() fails, rpcPos points to the source of the error. URIs do not skip trailing LWS.

Returns

resSI_SIPPARSER_DATACONT : CAbsoluteUri (see page 128) body found, more data follows.

resS_OK : CAbsoluteUri (see page 128) body found, end of data follows.

resFE_UNEXPECTED : No body could be found after the colon.

Description

Parses an unknown URI type's body. Only copies all legal characters into the m_strBody member. Note that this method does not explicitly parse the (hier-part / opaque-part).

The scheme must be set manually after Parse() is called. CAbsoluteUri (see page 128) does not handle the parsing of the scheme, it only serves as a container for that data (it serializes the scheme however).

See Also

GetUriLength, SetScheme (see page 133).

3.4.1.1.3.7 - CAbsoluteUri::Reset Method

Reinitializes the instance.

C++

```
virtual void Reset();
```

Description

Sets the scheme and body to empty values. It does not reset the bracket enclosing setting.

3.4.1.1.3.8 - CAbsoluteUri::Serialize Method

Outputs the data member in a format that is ready to be sent on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Where to output the internal data.

Description

Outputs the scheme and the body. Both scheme and body are output as they are, without escaping.

3.4.1.1.3.9 - CAbsoluteUri::SetScheme Method

Sets the scheme part of the URI.

C++

```
void SetScheme(IN const CString& rstrScheme);
```

Parameters

Parameters	Description
IN const CString& rstrScheme	Scheme to set.

Description

Sets the scheme to use.

See Also

GetScheme (see page 131)

3.4.1.1.4 - Operators

3.4.1.1.4.1 - CAbsoluteUri::= Operator

Assignment operator.

C++

```
CAbsoluteUri& operator =(IN const CAbsoluteUri& rSrc);
```

Parameters

Parameters	Description
IN const CAbsoluteUri& rSrc	CAbsoluteUri (see page 128) from which to copy.

Returns

This instance of CAbsoluteUri (see page 128).

Description

Default assignment operator.

3.4.1.2 - CDate Class

Class Hierarchy

```
CDate
```

C++

```
class CDate;
```

Description

This class represents the value of a Date header field. It corresponds to the SIP-date construct of the RFC 3261 BNF.

RFC 3261 BNF:

```

Date      = "Date" HCOLON SIP-date
SIP-date  = rfc1123-date
rfc1123-date = wkday "," SP date1 SP time SP "GMT"
date1    = 2DIGIT SP month SP 4DIGIT
           ; day month year (e.g., 02 Jun 1982)
time     = 2DIGIT ":" 2DIGIT ":" 2DIGIT
           ; 00:00:00 - 23:59:59
wkday    = "Mon" / "Tue" / "Wed"
           / "Thu" / "Fri" / "Sat" / "Sun"
month    = "Jan" / "Feb" / "Mar" / "Apr"

```

```

/ "May" / "Jun" / "Jul" / "Aug"
/ "Sep" / "Oct" / "Nov" / "Dec"

```

Location

SipParser/CDate.h

Constructors

Constructor	Description
◆ CDate (See page 136)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
◆ ~CDate (See page 136)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
◆ != (See page 139)	Comparison operator.
◆ = (See page 139)	Assignment operator.
◆ == (See page 140)	Comparison operator.

Legend

	Method
--	--------

Methods

Method	Description
◆ GetDateTime (See page 136)	Provides access to the SDateTime (See page 135) structure.
◆ GetDayOfMonth (See page 136)	Gets the day.
◆ GetDayOfWeek (See page 137)	Gets the day of the week.
◆ GetHours (See page 137)	Gets the hours.
◆ GetMinutes (See page 137)	Gets the minutes.
◆ GetMonth (See page 137)	Gets the month.
◆ GetSeconds (See page 138)	Gets the seconds.
◆ GetYear (See page 138)	Gets the year.
◆ IsSet (See page 138)	Returns true if the date is already set.
◆ Parse (See page 138)	Parses the date.
◆ Reset (See page 139)	Resets date members to initial state.
◆ ~CDate (See page 139)	Outputs the date.

Legend

	Method
	virtual

Structs

Struct	Description
SDDateTime (See page 135)	

3.4.1.2.1 - Data Members

3.4.1.2.1.1 - CDate::ms_szDATE_GMT Data Member

```
const char* const ms_szDATE_GMT;
```

Description

Contains the name of the GMT time zone.

3.4.1.2.1.2 - CDate::ms_szDAY_SUNDAY Data Member

```
const char* const ms_szDAY_SUNDAY;
const char* const ms_szDAY_MONDAY;
const char* const ms_szDAY_TUESDAY;
const char* const ms_szDAY_WEDNESDAY;
const char* const ms_szDAY_THURSDAY;
const char* const ms_szDAY_FRIDAY;
const char* const ms_szDAY_SATURDAY;
```

Description

Constant containing the day's name.

3.4.1.2.1.3 - CDate::ms_szMONTH_JANUARY Data Member

```
const char* const ms_szMONTH_JANUARY;
const char* const ms_szMONTH_FEBRUARY;
const char* const ms_szMONTH_MARCH;
const char* const ms_szMONTH_APRIIL;
const char* const ms_szMONTH_MAY;
const char* const ms_szMONTH_JUNE;
const char* const ms_szMONTH_JULY;
const char* const ms_szMONTH_AUGUST;
const char* const ms_szMONTH_SEPTEMBER;
const char* const ms_szMONTH_OCTOBER;
const char* const ms_szMONTH_NOVEMBER;
const char* const ms_szMONTH_DECEMBER;
```

Description

Constant containing the month's name.

3.4.1.2.2 - Structs

3.4.1.2.2.1 - CDate::SDateTime Struct

```
struct SDateTime {
    uint8_t m_uSec;
    uint8_t m_uMin;
    uint8_t m_uHour;
    uint8_t m_uDay;
    uint8_t m_uMonth;
    uint16_t m_uYear;
    uint8_t m_uDayOfWeek;
};
```

Description

Structure that contains a date and time.

Members

Members	Description
uint8_t m_uSec;	Seconds after the minute - [0,59].
uint8_t m_uMin;	Minutes after the hour - [0,59].
uint8_t m_uHour;	Hours since midnight - [0,23].
uint8_t m_uDay;	Day of the month - [1,31].
uint8_t m_uMonth;	Months since January - [0,11].

<code>uint16_t m_uYear;</code>	Years.
<code>uint8_t m_uDayOfWeek;</code>	Days since Sunday - [0,6].

3.4.1.2.3 - Constructors

3.4.1.2.3.1 - CDate::CDate Constructor

Constructor.

C++

```
CDate();
```

Description

Constructor.

3.4.1.2.4 - Destructors

3.4.1.2.4.1 - CDate::~CDate Destructor

Destructor.

C++

```
virtual ~CDate();
```

Description

destructor.

3.4.1.2.5 - Methods

3.4.1.2.5.1 - GetDateTime

3.4.1.2.5.1.1 - CDate::GetDateTime Method

Provides access to the SDDateTime (see page 135) structure.

C++

```
const SDDateTime& GetDateTime() const;
SDDateTime& GetDateTime();
```

Returns

SDDateTime (see page 135) A reference on the date time structure.

Description

Provides access to the SDDateTime (see page 135) structure.

3.4.1.2.5.2 - CDate::GetDayOfMonth Method

Gets the day.

C++

```
uint8_t GetDayOfMonth() const;
```

Returns

The day of the month value.

Description

Returns the day of the month (1 to 31).

3.4.1.2.5.3 - CDate::GetDayOfWeek Method

Gets the day of the week.

C++

```
CTime::EDayOfWeek GetDayOfWeek() const;
```

Returns

The day of the week value.

Description

Returns the day of the week.

3.4.1.2.5.4 - CDate::GetHours Method

Gets the hours.

C++

```
uint8_t GetHours() const;
```

Returns

The hours value.

Description

Returns the hours.

3.4.1.2.5.5 - CDate::GetMinutes Method

Gets the minutes.

C++

```
uint8_t GetMinutes() const;
```

Returns

The minutes value.

Description

Returns the minutes.

3.4.1.2.5.6 - CDate::GetMonth Method

Gets the month.

C++

```
uint8_t GetMonth() const;
```

Returns

The month value (1 to 12).

Description

Returns the month.

3.4.1.2.5.7 - CDate::GetSeconds Method

Gets the seconds.

C++

```
uint8_t GetSeconds() const;
```

Returns

The seconds value.

Description

Returns the seconds.

3.4.1.2.5.8 - CDate::GetYear Method

Gets the year.

C++

```
uint16_t GetYear() const;
```

Returns

The year value.

Description

Returns the year.

3.4.1.2.5.9 - CDate::IsSet Method

Returns true if the date is already set.

C++

```
const bool IsSet() const;
```

Returns

True if the date is already set.

Description

Checks if the date has been set.

3.4.1.2.5.10 - CDate::Parse Method

Parses the date.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data. It is adjusted as parsing progresses.

Returns

resFE_SIPPARSER_TOKEN_NOT_FOUND One of the required parameter has not been found.

resFE_INVALID_ARGUMENT : The data to parse indicates an empty header, but the header type cannot be empty.

resSI_SIPPARSER_DATACONT : The header has been parsed, more data follows.

resS_OK : The header has been parsed, end of data follows.

Description

This method parses a date header. If a separating comma is reached, parsing stops.

3.4.1.2.5.11 - CDate::Reset Method

Resets date members to initial state.

C++

```
void Reset();
```

Description

This resets the date to its initial state.

3.4.1.2.5.12 - CDate::Serialize Method

Outputs the date.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Where to output the internal data.

Description

Outputs the date data as per the SIP-date construct of the RFC 3261 BNF.

3.4.1.2.6 - Operators**3.4.1.2.6.1 - CDate::!= Operator**

Comparison operator.

C++

```
bool operator !=(IN const CDate& rSrc) const;
```

Parameters

Parameters	Description
IN const CDate& rSrc	Source against which to compare.

Returns

True if the addr-spec parts are not equivalent.

Description

Comparison operator. Compares the SDaTeTime (see page 135) struct member.

3.4.1.2.6.2 - CDate::= Operator

Assignment operator.

C++

```
CDate& operator =(const CDate& rSrc);
```

Parameters

Parameters	Description
const CDate& rSrc	Source from which to copy.

Returns

A reference on "this" instance.

Description

Assignment operator.

3.4.1.2.6.3 - CDate::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CDate& rSrc) const;
```

Parameters

Parameters	Description
IN const CDate& rSrc	Source against which to compare.

Returns

True if the date parts are equivalent.

Description

Comparison operator. Compares the tm struct member.

3.4.1.3 - CGenericParam Class

Class: CGenericParam

Class Hierarchy

```
CGenericParam
```

C++

```
class CGenericParam;
```

Description

This class abstracts all SIP parameters. Currently three charsets are supported: eCS_SIPHEADER_PARAM, eCS_SIPURI_PARAM, and eCS_TELURI_PARAM.

The parameters are implemented by using CTokens. The name part of the all parameters types is always a token, and the value part is either a token, a host, and in the case of eCS_SIP_HEADER, may be a quoted string.

Note that the name and value members are tokens and should be mapped to the same character set. Otherwise, the behaviour is undefined.

Refer to the CToken (see page 360) documentation for an overview of the charsets.

```
RFC 3261 ABNF
For eCS_SIP_HEADER:
generic-param = token [ EQUAL gen-value ]
gen-value     = token / host / quoted-string

For eCS_SIPURI_PARAM:
other-param   = pname [ "=" pvalue ]
pname         = 1*paramchar
pvalue        = 1*paramchar

For eCS_TELURI_PARAM:
par           = parameter / extension / isdn-subaddress
isdn-subaddress = ";isub=" 1*uric
```

```

extension          = ";ext=" 1*phonedigit
context            = ";phone-context=" descriptor
parameter          = ";" pname [= pvalue ]
                    = 1*( alphanum / "-")
                    = 1*paramchar
paramchar          = param-unreserved / unreserved / pct-encoded
unreserved         = alphanum / mark
                    = "-" / "_" / "." / "!" / "~" / "*" /
                    = " " / "(" / ")"
pct-encoded        = "%" HEXDIG HEXDIG
param-unreserved  = "[" / "]" / "/" / ":" / "&" / "+" / "$"
phonedigit         = DIGIT / [ visual-separator ]
phonedigit-hex    = HEXDIG / "*" / "#" / [ visual-separator ]
visual-separator   = "-" / "." / "(" / ")"
alphanum           = ALPHA / DIGIT
reserved           = ";" / "/" / "?" / ":" / "@" / "&" /
                    = "=" / "+" / "$" / ","
uric               = reserved / unreserved / pct-encoded

```

Location

SipParser/CGenericParam.h

See Also

CToken (see page 360)

Constructors

Constructor	Description
CGenericParam (see page 142)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~CGenericParam (see page 143)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 145)	Inequality Operator.
 = (see page 145)	Assignment Operator.
 == (see page 146)	Comparison Operator.

Legend

	Method
--	--------

Methods

Method	Description
 GetName (see page 143)	Provides access to the parameter's name.
 GetValue (see page 143)	Provides access to the parameter's value.
 Parse (see page 144)	Parses the parameter, including name and optional value.
 Reset (see page 144)	Resets this object.
 Serialize (see page 144)	Outputs the parameter. It outputs the value if non-empty. It outputs the escaped characters if the token type supports it.
 ViaBranchStartsWithMagicCookie (see page 145)	Returns true if the parameter's value starts with the branch magic cookie.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
ECharSet (see page 146)	

3.4.1.3.1 - Constructors

3.4.1.3.1.1 - CGenericParam

3.4.1.3.1.1.1 - CGenericParam::CGenericParam Constructor

Constructor.

C++

```
CGenericParam(IN ECharSet eCharSet);
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set that this parameter supports.

Description

Extended constructor.

3.4.1.3.1.1.2 - CGenericParam::CGenericParam Constructor

Constructor.

C++

```
CGenericParam(IN ECharSet eCharSet, IN const CString& rstrName, IN const CString& rstrValue);
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set that this parameter supports.
IN const CString& rstrName	Name.
IN const CString& rstrValue	Value.

Description

Extended constructor. No validation is made on the name and value.

3.4.1.3.1.1.3 - CGenericParam::CGenericParam Constructor

Constructor.

C++

```
CGenericParam(IN const CGenericParam& rSrc);
```

Parameters

Parameters	Description
IN const CGenericParam& rSrc	Parameter to copy.

Description

Copy constructor.

3.4.1.3.1.1.4 - CGenericParam::CGenericParam Constructor

Constructor.

C++

```
CGenericParam( IN const CToken& rTokName, IN const CToken& rTokValue);
```

Parameters

Parameters	Description
IN const CToken& rTokName	Name.
IN const CToken& rTokValue	Value.

Description

Extended constructor.

3.4.1.3.2 - Destructors**3.4.1.3.2.1 - CGenericParam::~CGenericParam Destructor**

Destructor.

C++

```
virtual ~CGenericParam();
```

Description

Destructor.

3.4.1.3.3 - Methods**3.4.1.3.3.1 - GetName****3.4.1.3.3.1.1 - CGenericParam::GetName Method**

Provides access to the parameter's name.

C++

```
const CToken& GetName() const;
CToken& GetName();
```

Returns

Name of the parameter.

Description

Provides access to the parameter name.

See Also

GetValue (see page 143)

3.4.1.3.3.2 - GetValue**3.4.1.3.3.2.1 - CGenericParam::GetValue Method**

Provides access to the parameter's value.

C++

```
const CToken& GetValue() const;
CToken& GetValue();
```

Returns

Value of the parameter.

Description

Provides access to the parameter value.

See Also

GetName (see page 143)

3.4.1.3.3.3 - CGenericParam::Parse Method

Parses the parameter, including name and optional value.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. CGenericParam (see page 140) automatically advances rpcPos over any LWS encountered AFTER parsing its data. If the parsing fails while parsing the name, rpcPos is left untouched. If the parsing fails while looking for the value, rpcPos is set to the value's start position after the EQUAL.

Returns

resSI_SIPPARSER_DATACONT : Parameter found, optional LWS skipped (only applies when the token charset allows LWS), more data follows. The optional value may have been found.

resS_OK : Parameter found, optional LWS skipped (only applies when the token charset allows LWS), end of data follows. The optional value may have been found.

resFE_INVALID_ARGUMENT : May only be returned for the eCS_SIP_HEADER charset, this code means that the value part looked like a quoted string but its end could not be found.

resFE_UNEXPECTED : Parameter name has not been found.

resFE_UNEXPECTED : Parameter name and EQUAL have been found, but no value could be found.

Description

Parses a parameter, advancing rpcPos as it goes. First, the name is parsed (a token). Then, if the optional EQUAL is found, it means that a value MUST be present. The value is then parsed, and can be either a token, host, or quoted-string.

3.4.1.3.3.4 - CGenericParam::Reset Method

Resets this object.

C++

```
void Reset();
```

Description

Clears the internal data.

3.4.1.3.3.5 - CGenericParam::Serialize Method

Outputs the parameter. It outputs the value if non-empty. It outputs the escaped characters if the token type supports it.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT Cblob& rBlob	Where to output the internal data.

Description

Outputs the parameter's data. It outputs something only if the name is not empty. If the value is not empty, it also adds an EQUAL after outputting the name and outputs the value. If the token's charset supports it, illegal characters are output in their escaped form.

3.4.1.3.3.6 - CGenericParam::ViaBranchStartsWithMagicCookie Method

Returns true if the parameter's value starts with the branch magic cookie.

C++

```
bool ViaBranchStartsWithMagicCookie() const;
```

Returns

Returns true if the parameter's value starts with the branch magic cookie.

Description

This method checks if the value starts with the magic cookie "z9hG4bK" as per RFC 3261 section 8.1.1.7. This method is case-insensitive.

Example

```
z9hG4bKMoreData starts with the magic cookie.
Z9Hg4BkMoreData starts with the magic cookie.
z9hG4bK does not start with the magic cookie (must have more data).
```

3.4.1.3.4 - Operators**3.4.1.3.4.1 - CGenericParam::!= Operator**

Inequality Operator.

C++

```
bool operator !=(IN const CGenericParam& rSrc) const;
```

Parameters

Parameters	Description
IN const CGenericParam& rSrc	Parameter against which to compare.

Returns

True if the parameter names and/or values are not equal.

Description

Inequality operator.

3.4.1.3.4.2 - CGenericParam::= Operator

Assignment Operator.

C++

```
CGenericParam& operator =(IN const CGenericParam& rSrc);
```

Parameters

Parameters	Description
IN const CGenericParam& rSrc	Parameter from which to copy.

Returns

Returns this CGenericParam (see page 140) instance.

Description

Default assignment operator. Copies the name and value.

3.4.1.3.4.3 - CGenericParam::== Operator

Comparison Operator.

C++

```
bool operator ==(IN const CGenericParam& rSrc) const;
```

Parameters

Parameters	Description
IN const CGenericParam& rSrc	Parameter against which to compare.

Returns

True if the parameter names and values are equal.

Description

Comparison operator. The name and value are compared, relative to the charset's case sensitivity.

3.4.1.3.5 - Enumerations**3.4.1.3.5.1 - CGenericParam::ECharSet Enumeration**

```
enum ECharSet {
    eCS_SIP_HEADER = CToken::eCS_SIPHEADER_PARAM,
    eCS_SIPURI_PARAM = CToken::eCS_SIPURI_PARAM,
    eCS_TELURI_PARAM = CToken::eCS_TELURI_PARAM
};
```

Description

Indicate which character set to use for the CGenericParam (see page 140).

Members

Members	Description
eCS_SIP_HEADER = CToken::eCS_SIPHEADER_PARAM	See CToken::ECharSet (see page 378)
eCS_SIPURI_PARAM = CToken::eCS_SIPURI_PARAM	See CToken::ECharSet (see page 378)
eCS_TELURI_PARAM = CToken::eCS_TELURI_PARAM	See CToken::ECharSet (see page 378)

3.4.1.4 - CGenParamList Class**Class Hierarchy**

```
CGenParamList
```

C++

```
class CGenParamList;
```

Description

This class implements a list of SIP parameters, separated by a specified character. The CGenParamList can handle any type of CGenericParam (see page 140).

Warning

This class' algorithm uses the premise that all parameter names are unique. The application should not add two parameters that have the same name.

```
RFC 3261 ABNF
(param-list)  =  *( SEMI generic-param )
generic-param =  token [ EQUAL gen-value ]
gen-value     =  token / host / quoted-string
```

Location

SipParser/CGenParamList.h

See Also

CGenericParam (see page 140)

Example

Fetching a parameter named "tag".

```
CGenParamList paramList;

// I already know that the first parameter in the list is "tag".
//-----
CGenericParam* pTagParam1 = paramList[0];

if(pTagParam1 != NULL)
{
    // The list contained at least one parameter.
    //-----
    if(pTagParam1->GetName() == "tag")
    {
        // Check the value.
        //-----
        if(pTagParam1->GetValue() == "some tag")
        {
            ...
        }
    }
    else
    {
        // Not the param i'm looking for.
        //-----
    }
}
else
{
    // tag param was not found
}

// I'd like the list to find the parameter by its name.
//-----
CGenericParam* pTagParam2 = paramList["tag"];

if(pTagParam2 != NULL)
{
    // tag param was found
}
else
{
    // tag param was not found
}
```

Constructors

Constructor	Description
CGenericParamList (see page 148)	Constructor.

Legend

	Constructor
	Method

Destructors

Destructor	Description
~CGenericParamList (see page 149)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 152)	Comparison operator. Compares the parameter lists following RFC 3261 rules. For SipUri parameters, the rules are those of section 19.1.4. For SIP headers, all parameters must be present and match.
 [] (see page 152)	Returns the parameter whose name matches.
 = (see page 153)	Assignment operator.
 == (see page 154)	Comparison operator. Compares the parameter lists following RFC 3261 rules. For SipUri parameters, the rules are those of section 19.1.4. For SIP headers, all parameters must be present and match.

Legend

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

Methods

Method	Description
 Append (see page 149)	Adds a parameter at the end of the list. Does not check if the parameter name is already in the list.
 IsEmpty (see page 149)	Returns true if the list contains no parameter.
 Length (see page 150)	Returns the number of parameters in the list.
 Parse (see page 150)	Parses the parameters list beginning at rpcPos. cSeparator is the character that separates the parameters.
 Remove (see page 150)	Removes all parameters of the same name from the list.
 Reset (see page 151)	Resets this object.
 Serialize (see page 151)	Inserts cSeparator after each parameter, except for the last parameter.
 Set (see page 152)	Replaces or Adds the parameter.

Legend

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

Enumerations

Enumeration	Description
 EStartWithSeparator (see page 154)	Enum to use in parameter to Parse (see page 150)().

3.4.1.4.1 - Constructors

3.4.1.4.1.1 - CGenParamList

3.4.1.4.1.1.1 - CGenParamList::CGenParamList Constructor

Constructor.

C++

```
CGenParamList();
```

Description

Default constructor.

3.4.1.4.1.1.2 - CGenParamList::CGenParamList Constructor

Copy Constructor.

C++

```
CGenParamList(IN const CGenParamList& rSrc);
```

Parameters

Parameters	Description
IN const CGenParamList& rSrc	List to copy.

Description

Copy constructor.

3.4.1.4.2 - Destructors**3.4.1.4.2.1 - CGenParamList::~CGenParamList Destructor**

Destructor.

C++

```
virtual ~CGenParamList();
```

Description

Destructor.

3.4.1.4.3 - Methods**3.4.1.4.3.1 - CGenParamList::Append Method**

Adds a parameter at the end of the list. Does not check if the parameter name is already in the list.

C++

```
unsigned int Append(IN TO CGenericParam* pParam);
```

Parameters

Parameters	Description
IN TO CGenericParam* pParam	Parameter to add to the list. Ownership is taken.

Returns

The number of parameters in the list including the new addition.

Description

This method adds a parameter at the end of the list. It does not verify if the parameter name is unique in the list.

3.4.1.4.3.2 - CGenParamList::IsEmpty Method

Returns true if the list contains no parameter.

C++

```
bool IsEmpty() const;
```

Returns

True if the parameter list contains no parameter.

Description

Returns true if the parameter list contains no parameter.

3.4.1.4.3.3 - CGenParamList::Length Method

Returns the number of parameters in the list.

C++

```
unsigned int Length() const;
```

Returns

The number of parameters in the list.

Description

Returns the number of parameters in the list.

3.4.1.4.3.4 - CGenParamList::Parse Method

Parses the parameters list beginning at rpcPos. cSeparator is the character that separates the parameters.

C++

```
mxt_result Parse(IN CGenericParam::ECharSet eCharSet, INOUT const char*& rpcPos, IN EStartWithSeparator eStartWithSep = eWITH_SEP, IN char cSeparator = ';');
```

Parameters

Parameters	Description
IN CGenericParam::ECharSet eCharSet	Character set to use while parsing. The generated CGenericParam (see page 140) uses that character set and is stored in the list. This also affects later serialization and comparison if the same CGenericParam (see page 140) instances are reused.
INOUT const char*& rpcPos	Pointer to the start of the parameter list. It either points to the first separator or the start of the first parameter's name, depending on the value of eStartWithSep. If the method fails, rpcPos is set on the position of the error. If the method succeeds, rpcPos is set after the last parsed parameter.
IN EStartWithSeparator eStartWithSep = eWITH_SEP	If true, parsing assumes that the cSeparator character should be found when parsing starts. In other words, if true, rpcPos should initially point to a cSeparator character, followed by the parameters. If false, parsing assumes that rpcPos initially points to the start of a parameter name.
IN char cSeparator = ';'	The character that separates parameters in the list.

Returns

resSI_SIPPARSER_DATACONT : The parameter list has been found and at least one parameter has been correctly parsed. The optional LWS has been skipped (only applies when the token charset allows LWS), more data follows.

resS_OK : The parameter list has been found and at least one parameter has been correctly parsed. The optional LWS has been skipped (only applies when the token charset allows LWS), end of data follows.

resFE_UNEXPECTED : No parameters has been found.

May also return any mxt_result the CGenericParam::Parse (see page 144) may return.

Description

This method iteratively parses a list of generic parameters separated by a specified character. This method allocates CGenericParam (see page 140) on the heap and uses them to parse each parameter. The parsed parameters then populate the m_vecpParam member.

When the method succeeds, rpcPos points past the parameters and any trailing LWS. When the method fails, the same rules than for CGenericParam (see page 140) parsing apply for rpcPos' position. When the method fails, rpcPos is not reset to the initial position.

3.4.1.4.3.5 - Remove

3.4.1.4.3.5.1 - CGenParamList::Remove Method

Removes all parameters of the same name from the list.

C++

```
bool Remove(IN const CString& rstrName);
```

Parameters

Parameters	Description
IN const CString& rstrName	Name of the parameter to remove.

Returns

True if the parameter has been correctly removed.

Description

Removes all parameters of the same name from the list.

3.4.1.4.3.5.2 - CGenParamList::Remove Method

Removes the parameter at the specified index.

C++

```
bool Remove(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	Zero-based index of the parameter to remove.

Returns

True if the parameter has been correctly removed.

Description

Removes a parameter from the list.

3.4.1.4.3.6 - CGenParamList::Reset Method

Resets this object.

C++

```
void Reset();
```

Description

Clears the parameter list.

3.4.1.4.3.7 - CGenParamList::Serialize Method

Inserts cSeparator after each parameter, except for the last parameter.

C++

```
void Serialize(INOUT CBlob& rBlob, IN char cSeparator = ';') const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Where to output the parameter list.
IN char cSeparator = ','	Separator to output between parameters.

Description

Outputs all parameters in the list. The output starts with the first parameter (no separator is output before the first parameter). No separator is added after the last parameter.

3.4.1.4.3.8 - CGenParamList::Set Method

Replaces or Adds the parameter.

C++

```
void Set(IN TO CGenericParam* pParam);
```

Parameters

Parameters	Description
IN TO CGenericParam* pParam	Parameter to set. Ownership is taken.

Description

Replaces the parameter if the name is found, otherwise appends the parameter to the list. If multiple parameters have the same name, only the first is updated.

3.4.1.4.4 - Operators

3.4.1.4.4.1 - CGenParamList::!= Operator

Comparison operator. Compares the parameter lists following RFC 3261 rules. For SipUri parameters, the rules are those of section 19.1.4. For SIP headers, all parameters must be present and match.

C++

```
bool operator !=(IN const CGenParamList& rSrc) const;
```

Parameters

Parameters	Description
IN const CGenParamList& rSrc	List against which to compare.

Returns

True if the lists are not equivalent following RFC 3261 rules.

Description

Inequality operator. See CGenParamList::operator==.

See Also

operator==

3.4.1.4.4.2 - []

3.4.1.4.4.2.1 - CGenParamList::[] Operator

Returns the parameter whose name matches.

C++

```
inline CGenericParam* operator [](IN const CString& rstrName);
inline const CGenericParam* operator [](IN const CString& rstrName) const;
```

Parameters

Parameters	Description
IN const CString& rstrName	Name of the parameter to fetch. Case sensitivity depends on the type of parameter this list is holding.
IN const CString& rstrName	Name of the parameter to fetch. Case sensitivity depends on the type of parameter this list is holding.

Returns

Pointer to the first parameter that matches the name. It is NULL if the name is not matched. Note that if two parameters have the same name, only the first one is returned.

Description

This method provides array-style access to individual parameters in the list.

3.4.1.4.4.2.2 - CGenParamList::[] Operator

Returns the parameter whose name matches.

C++

```
inline CGenericParam* operator [](IN const char* pszName);
inline const CGenericParam* operator [](IN const char* pszName) const;
```

Parameters

Parameters	Description
IN const char* pszName	Name of the parameter to fetch. Case sensitivity depends on the type of parameter this list is holding.
IN const char* pszName	Name of the parameter to fetch. Case sensitivity depends on the type of parameter this list is holding.

Returns

Pointer to the first parameter that matches the name. It is NULL if the name is not matched. Note that if two parameters have the same name, only the first one is returned.

Description

This method provides array-style access to individual parameters in the list.

3.4.1.4.4.2.3 - CGenParamList::[] Operator

Returns the parameter at uIndex.

C++

```
inline CGenericParam* operator [](IN unsigned int uIndex);
inline const CGenericParam* operator [](IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	Zero-based index of the parameter to fetch.
IN unsigned int uIndex	Zero-based index of the parameter to fetch.

Returns

Pointer to the parameter at the given index. NULL if the index is out-of-bounds.

Description

This method provides array-style access to individual parameters in the list.

3.4.1.4.4.3 - CGenParamList:::= Operator

Assignment operator.

C++

```
CGenParamList& operator =(IN const CGenParamList& rSrc);
```

Parameters

Parameters	Description
IN const CGenParamList& rSrc	List to copy.

Returns

Returns this instance.

Description

Default assignment operator.

3.4.1.4.4.4 - CGenParamList::== Operator

Comparison operator. Compares the parameter lists following RFC 3261 rules. For SipUri parameters, the rules are those of section 19.1.4. For SIP headers, all parameters must be present and match.

C++

```
bool operator ==(IN const CGenParamList& rSrc) const;
```

Parameters

Parameters	Description
IN const CGenParamList& rSrc	List against which to compare.

Returns

True if the lists are equivalent as per RFC 3261.

Description

Comparison operator. Compares the parameter lists using RFC 3261 rules.

The Sip header parameter list comparison uses the following rules:

- All parameters must be present in both lists.
- All parameter values must match.

The SipUri parameter list comparison uses the following rules:

- Parameters that appear in both lists must match.
- The user, method, maddr, and ttl parameters must match if present in any of the lists.
- Other parameters appearing in only one list are ignored.

See Also

CompareUriParamList

3.4.1.4.5 - Enumerations**3.4.1.4.5.1 - CGenParamList::EStartWithSeparator Enumeration**

Enum to use in parameter to Parse (see page 150)().

C++

```
enum EStartWithSeparator {
    eNO_SEP,
    eWITH_SEP
};
```

Members

Members	Description
eNO_SEP	Parse (see page 150) begins with a parameter name.
eWITH_SEP	Parse (see page 150) begins with a separator.

3.4.1.5 - CHeaderList Class

Class Hierarchy



C++

```
class CHeaderList;
```

Description

This class manages a list of headers. It is responsible for parsing the headers when required. It serves as a container for the headers. It also handles the caching of headers and serializes multi-headers.

Location

SipParser/CHeaderList.h

See Also

CSipHeader (see page 239)

Constructors

Constructor	Description
 CHeaderList (see page 156)	Default Constructor.

Legend

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

Destructors

Destructor	Description
  ~CHeaderList (see page 156)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 166)	Inequality operator.
 = (see page 166)	Assignment operator.
 == (see page 166)	Comparison operator.

Legend

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

Methods

Method	Description
 Append (see page 157)	Adds the header list at the end of the list.
 AppendRawData (see page 157)	Buffers received data. Allows to parse headers without requiring a start-line. Input must be a NULL-terminated string.
 CommitRawDataList (see page 158)	Instructs to use its list of buffered raw headers and build the CHeaderList with them.
 Get (see page 159)	Returns a parsed version of a CSipHeader (see page 239) chain.
 GetNbHeaderTypes (see page 161)	Returns the number of header types that are available in the list.
 GetRawDataList (see page 162)	Provides access to the list of buffered headers.
 IsEmpty (see page 162)	Returns true if the list contains nothing.
 Prepend (see page 162)	Inserts the header at the start of the list.
 RemoveHeader (see page 163)	Removes the n'th header of the specified type.
 RemoveHeaderType (see page 163)	Removes all headers of the specified type.

• ReplaceHeaders (see page 164)	Appends the header list and replaces the headers that have the same type as the one in the passed list.
• ReplaceHeaderTypeWith (see page 164)	Replaces all the headers of a type with a new header.
• Reset (see page 165)	Resets the header list to its initial state.
• Serialize (see page 165)	Outputs the headers.
• Sort (see page 165)	Sorts the headers using the defined header order.

Legend

•	Method
---	--------

3.4.1.5.1 - Constructors**3.4.1.5.1.1 - CHeaderList****3.4.1.5.1.1.1 - CHeaderList::CHeaderList Constructor**

Default Constructor.

C++

```
CHeaderList();
```

Description

Constructor.

3.4.1.5.1.1.2 - CHeaderList::CHeaderList Constructor

Copy Constructor.

C++

```
CHeaderList(IN const CHeaderList& rSrc);
```

Parameters

Parameters	Description
IN const CHeaderList& rSrc	Header list from which to initialize.

Description

Copy constructor.

3.4.1.5.2 - Destructors**3.4.1.5.2.1 - CHeaderList::~CHeaderList Destructor**

Destructor.

C++

```
virtual ~CHeaderList();
```

Description

Destructor.

3.4.1.5.3 - Methods**3.4.1.5.3.1 - Append**

3.4.1.5.3.1.1 - CHeaderList::Append Method

Adds the header list at the end of the list.

C++

```
mxt_result Append(IN TO CHeaderList* pNewHeaderList);
```

Parameters

Parameters	Description
IN TO CHeaderList* pNewHeaderList	Header list to add in the list. The headers are taken from that list and appended into this list. Ownership is TAKEN.

Returns

See CHeaderList::Insert().

Description

Adds the given header list at the end of the current header list. All headers are added at the end of the list, except if other headers of the same type are present, in which case the new header is appended to the last of these. The append mechanism uses the CSipHeader (see page 239)'s next header feature.

See Also

Insert

3.4.1.5.3.1.2 - CHeaderList::Append Method

Adds the header at the end of the list.

C++

```
mxt_result Append(IN TO CSipHeader* pNewHeader, IN bool bTakeOwnershipOnSuccess = false);
```

Parameters

Parameters	Description
IN TO CSipHeader* pNewHeader IN bool bTakeOwnershipOnSuccess = false	Header to add in the list. • false: Default value. Ownership of pNewHeader is always taken. • true: Ownership is taken if the return value is a success only.

Returns

See CHeaderList::Insert().

Description

Adds the given header in the header list. The header is added at the end of the list, except if other headers of the same type are present, in which case the new header is appended to the last of these. The append mechanism uses the CSipHeader (see page 239)'s next header feature.

See Also

Prepend (see page 162)

3.4.1.5.3.2 - CHeaderList::AppendRawData Method

Buffers received data. Allows to parse headers without requiring a start-line. Input must be a NULL-terminated string.

C++

```
mxt_result AppendRawData(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the received data. This string must be NULL-terminated. The pointer is adjusted as parsing progresses. Refer to the return codes for details of rpcPos's position after this method has completed.

Returns

resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF : All headers have been found. This packet is considered complete, but may also contain a payload that is not handled by this method. RpcPos points after the double CRLF.

resFE_UNEXPECTED : Error while parsing. The parser expected double CRLF to end *message-header. rpcPos points to the offending character.

Description

This method handles the buffering of the SIP packet's *message-header. Data received from the network can be buffered using this method. This method may be called multiple times, once each time new data is received on a connection.

This method also buffers each message-header into a CRawHeader (see page 218), and all CRawHeaders are put into a vector that can be retrieved by using the GetRawDataList (see page 162)() method.

The user of this method should append more data until either resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF or an error is returned.

Once the resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF result has been returned, the user is free to use the GetRawDataList (see page 162)() and CommitRawDataList (see page 158)() methods.

See Also

CRawHeader (see page 218), CommitRawDataList (see page 158), CSipPacketParser::AppendRawData (see page 314)

3.4.1.5.3.3 - CHeaderList::CommitRawDataList Method

Instructs to use its list of buffered raw headers and build the CHeaderList (see page 155) with them.

C++

```
mxt_result CommitRawDataList(OUT CVector<GO CSipHeader*>* pvecRefusedRawHeaders = NULL);
```

Parameters

Parameters	Description
pvecRefusedRawHeaders	Pointer to a vector of CSipHeaders. This is an OUT parameter. If this pointer is non-NULL, in the event that a raw header cannot be correctly committed, it is put into this header so that the caller can take action. Ownership of headers put into this vector is given to the caller.

Returns

resS_OK : Header list created.

resSW_SIPPARSER_SIPPACKET_COMMIT_COMPLETE_WITH_APPEND_ERRORS : Header list created, but errors have been encountered while appending headers. If pvecRefusedRawHeaders is non-NULL, checks the vector to see which headers caused this warning.

resFE_SIPPARSER_SIPPACKET_COMMIT_INVALID_STATE : There are no headers to commit.

resFE_INVALID_ARGUMENT : Error while parsing the status start line.

Description

This method converts the raw data (vector) list into a parsable CHeaderList (see page 155). This method must be called only after the AppendRawData (see page 157)() method has returned resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF. Once this method has succeeded, the user can use the CHeaderList (see page 155) directly. At that time, the raw data list is also removed from memory.

See Also

AppendRawData (see page 157), GetRawDataList (see page 162), CRawHeader (see page 218), CHeaderList (see page 155),

CSipPacketParser::CommitRawDataList (see page 315).

3.4.1.5.3.4 - Get

3.4.1.5.3.4.1 - CHeaderList::Get Method

Returns a parsed version of a CSipHeader (see page 239) chain.

C++

```
const CSipHeader* Get(IN ESipHeaderType eType, INOUT mxt_result* pres = NULL, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL) const;
CSipHeader* Get(IN ESipHeaderType eType, INOUT mxt_result* pres = NULL, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL);
```

Parameters

Parameters	Description
IN ESipHeaderType eType	Type of headers to parse and return.
INOUT mxt_result* pres = NULL	Pointer to a mxt_result where the result is set. It can be NULL if the result is not wanted. resFE_SIPPARSER_HEADERLIST_NOT_FOUND Could not find headers of the specified type. resFE_UNEXPECTED Headers of the eHDR_EXTENSION type are distinguished by their name. Use Get@IN const CString& rstrName@OUT const CSipHeader (see page 239)*& rpParsedHeader@IN unsigned int uMaxToParse instead. This method can also set pres to any value that the CSipHeader::ParseRawData() method may return. The most frequent are: resSI_SIPPARSER_DATACONT Success, uMaxToParse headers have been parsed, more headers are available. resS_OK Success, all available headers have been parsed. resSW_SIPPARSER_HEADER_NOT_ENOUGH_HEADERS Parsing did not encounter any error, but less than uMaxToParse headers are available.
IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL	Maximum number of times that the parse operation is launched on the CSipHeader (see page 239)'s next headers. This value is only an indication of the maximum effort that must be put forth by the CSipHeader (see page 239) before returning the list.

Returns

Reference on the pointer to the header list. It points on a CSipHeader (see page 239) chain if the method is successful.

Note that this method can return a NULL CSipHeader (see page 239) pointer even if the returned mxt_result is a success code. This happens if there is no more data to parse and the header has not been found. Also, a non NULL CSipHeader (see page 239) pointer can be returned even though the returned mxt_result is a failure code, i.e., this occurs when the header is present but the parsing failed.

Description

This method commands the parsing of a type of header. It is used to retrieve a list of headers of the specified type, in parsed format. Refer to CSipHeader::ParseRawData for more details on SIP header parsing.

See Also

CSipHeader::ParseRawData.

3.4.1.5.3.4.2 - CHeaderList::Get Method

Returns a parsed version of a CSipHeader (see page 239) chain.

C++

```
const CSipHeader* Get(IN ESipHeaderType eType, OUT mxt_result& rres, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL) const;
CSipHeader* Get(IN ESipHeaderType eType, OUT mxt_result& rres, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL);
```

Parameters

Parameters	Description
IN ESipHeaderType eType	Type of headers to parse and return.

OUT mxt_result& rres	Reference to a mxt_result where the result is set. resFE_SIPPARSER_HEADERLIST_NOT_FOUND Could not find headers of the specified type. resFE_UNEXPECTED Headers of the eHDR_EXTENSION type are distinguished by their name. Use Get@IN const CString& rstrName@OUT const CSipHeader (see page 239)*& rpParsedHeader@IN unsigned int uMaxToParse instead. This method can also set rres to any value that the CSipHeader::ParseRawData() method may return. The most frequent are: resSI_SIPPARSER_DATACONT Success, uMaxToParse headers have been parsed, more headers are available. resS_OK Success, all available headers have been parsed. resSW_SIPPARSER_HEADER_NOT_ENOUGH_HEADERS Parsing did not encounter any error, but less than uMaxToParse headers are available.
IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL	Maximum number of times that the parse operation is launched on the CSipHeader (see page 239)'s next headers. This value is only an indication of the maximum effort that must be put forth by the CSipHeader (see page 239) before returning the list.

Returns

Reference on the pointer to the header list. It points on a CSipHeader (see page 239) chain if the method is successful.

Note that this method can return a NULL CSipHeader (see page 239) pointer even if the returned mxt_result is a success code. This happens if there is no more data to parse and the header has not been found. Also, a non NULL CSipHeader (see page 239) pointer can be returned even though the returned mxt_result is a failure code, i.e., this occurs when the header is present but the parsing failed.

Description

This method commands the parsing of a type of header. It is used to retrieve a list of headers of the specified type, in parsed format. Refer to CSipHeader::ParseRawData for more details on SIP header parsing.

See Also

CSipHeader::ParseRawData.

3.4.1.5.3.4.3 - CHeaderList::Get Method

Returns a parsed version of a CSipHeader (see page 239) chain.

C++

```
const CSipHeader* Get(IN const CString& rstrName, INOUT mxt_result* pres = NULL, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL) const;
CSipHeader* Get(IN const CString& rstrName, INOUT mxt_result* pres = NULL, IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL);
```

Parameters

Parameters	Description
IN const CString& rstrName	Name of the headers to parse and return.
INOUT mxt_result* pres = NULL	Pointer to a mxt_result where the result is set. It can be NULL if the result is not wanted. resFE_SIPPARSER_HEADERLIST_NOT_FOUND Could not find headers of the specified type. resFE_UNEXPECTED The requested header name is invalid. This method can also be set to any value that the CSipHeader::ParseRawData() method may return. The most frequent are: resSI_SIPPARSER_DATACONT Success, uMaxToParse headers have been parsed, more headers are available. resS_OK Success, all available headers have been parsed. resSW_SIPPARSER_HEADER_NOT_ENOUGH_HEADERS Parsing did not encounter any error, but less than uMaxToParse headers are available.
IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL	Maximum number of times that the parse operation is launched on the CSipHeader (see page 239)'s next headers. This value is only an indication of the maximum effort that must be put forth by the CSipHeader (see page 239) before returning the list.

Returns

Reference on the pointer to the header list. It points on a CSipHeader (see page 239) chain if the method is successful.

Note that this method can return a NULL CSipHeader (see page 239) pointer even if the returned mxt_result is a success code. This happens if there is no more data to parse and the header has not been found. Also, a non NULL CSipHeader (see page 239) pointer can be returned even though the returned mxt_result is a failure code, i.e., this occurs when the header is present but the parsing failed.

Description

This method commands the parsing of a type of header. It is used to retrieve a list of headers of the specified type, in parsed format. Refer to CSipHeader::ParseRawData for more details on SIP header parsing.

See Also

CSipHeader::ParseRawData.

3.4.1.5.3.4.4 - CHeaderList::Get Method

Returns a parsed version of a CSipHeader (see page 239) chain.

C++

```
const CSipHeader* Get(IN const CString& rstrName, OUT mxt_result& rres, IN unsigned int uMaxToParse =
uHEADERS_PARSE_ALL) const;
CSipHeader* Get(IN const CString& rstrName, OUT mxt_result& rres, IN unsigned int uMaxToParse =
uHEADERS_PARSE_ALL);
```

Parameters

Parameters	Description
IN const CString& rstrName	Name of the headers to parse and return.
OUT mxt_result& rres	Reference to a mxt_result where the result is set. resFE_SIPPARSER_HEADERLIST_NOT_FOUND Could not find headers of the specified type. resFE_UNEXPECTED The requested header name is invalid. This method can also be set to any value that the CSipHeader::ParseRawData() method may return. The most frequent are: resSI_SIPPARSER_DATACONT Success, uMaxToParse headers have been parsed, more headers are available. resS_OK Success, all available headers have been parsed. resSW_SIPPARSER_HEADER_NOT_ENOUGH_HEADERS Parsing did not encounter any error, but less than uMaxToParse headers are available.
IN unsigned int uMaxToParse = uHEADERS_PARSE_ALL	Maximum number of times that the parse operation is launched on the CSipHeader (see page 239)'s next headers. This value is only an indication of the maximum effort that must be put forth by the CSipHeader (see page 239) before returning the list.

Returns

Reference on the pointer to the header list. It points on a CSipHeader (see page 239) chain if the method is successful.

Note that this method can return a NULL CSipHeader (see page 239) pointer even if the returned mxt_result is a success code. This happens if there is no more data to parse and the header has not been found. Also, a non NULL CSipHeader (see page 239) pointer can be returned even though the returned mxt_result is a failure code, i.e., this occurs when the header is present but the parsing failed.

Description

This method commands the parsing of a type of header. It is used to retrieve a list of headers of the specified type, in parsed format. Refer to CSipHeader::ParseRawData for more details on SIP header parsing.

See Also

CSipHeader::ParseRawData.

3.4.1.5.3.5 - CHeaderList::GetNbHeaderTypes Method

Returns the number of header types that are available in the list.

C++

```
unsigned int GetNbHeaderTypes() const;
```

Returns

Number of different header types.

Description

Counts the number of header types that are supported.

3.4.1.5.3.6 - CHeaderList::GetRawDataList Method

Provides access to the list of buffered headers.

C++

```
mxt_result GetRawDataList(OUT CVector<CRawHeader*>*& ppvecpRawData) const;
```

Parameters

Parameters	Description
OUT CVector<CRawHeader*>*& ppvecpRawData	Reference to the pointer that is updated if the call succeeds.

Returns

resSW_SIPPARSER_NULL_PTR : There is no raw data list.

resS_OK : There is a raw data list, the pointer is updated.

Description

Provides access to the raw data list.

See Also

AppendRawData (see page 157), CommitRawDataList (see page 158), CSipPacketParser::GetRawDataList (see page 316)

3.4.1.5.3.7 - CHeaderList::IsEmpty Method

Returns true if the list contains nothing.

C++

```
bool IsEmpty() const;
```

Returns

True if the list is empty.

Description

Returns true if the list is empty.

3.4.1.5.3.8 - CHeaderList::Prepend Method

Inserts the header at the start of the list.

C++

```
mxt_result Prepend(IN TO CSipHeader* pNewHeader, IN bool bTakeOwnershipOnSuccess = false);
```

Parameters

Parameters	Description
IN TO CSipHeader* pNewHeader	Header to add in the list.
IN bool bTakeOwnershipOnSuccess = false	<ul style="list-style-type: none"> false: Default value. Ownership of pNewHeader is always taken. true: Ownership is taken if the return value is a success only.

Returns

See `CHeaderList::Insert()`.

Description

Adds the given header in the header list. The header is inserted at the start of the list, except if other headers of the same type are present, in which case the new header is inserted at the start of these. The Prepend mechanism uses the `CSipHeader` (see page 239)'s next header feature.

See Also

`Prepend`

3.4.1.5.3.9 - RemoveHeader**3.4.1.5.3.9.1 - `CHeaderList::RemoveHeader` Method**

Removes the n'th header of the specified type.

C++

```
mxt_result RemoveHeader(IN ESipHeaderType eHeader, IN unsigned int uIndex);
mxt_result RemoveHeader(IN const CString& rstrName, IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN ESipHeaderType eHeader	Type of header to remove.
IN unsigned int uIndex	Index of the header to remove.
IN unsigned int uIndex	Index of the header to remove.
IN const CString& rstrName	Name of the header to remove.

Returns

`resS_OK` : Headers correctly removed.

`resFE_FAIL` : Could not find headers of the specified type or there is no header of that type, at that index. Note that headers of the `eHDR_EXTENSION` type must be removed with `Remove@IN const CString& rstrName`.

Description

Removes the n'th headers of the specified type.

3.4.1.5.3.10 - RemoveHeaderType**3.4.1.5.3.10.1 - `CHeaderList::RemoveHeaderType` Method**

Removes all headers of the specified type.

C++

```
mxt_result RemoveHeaderType(IN ESipHeaderType eHeader, OUT GO CSipHeader** ppSipHeader = NULL);
```

Parameters

Parameters	Description
IN ESipHeaderType eHeader	Type of header to remove.
OUT GO CSipHeader** ppSipHeader = NULL	If not NULL, the removed header is returned via this parameter. Ownership is GIVEN. The header can be NULL if not found.

Returns

`resS_OK` : Headers correctly removed.

`resFE_FAIL` : Could not find headers of the specified type. Note that headers of the `eHDR_EXTENSION` type must be removed with `Remove@IN const CString& rstrName`.

Description

Removes all headers of the specified type. If ppSipHeader is not NULL, ownership of the header is transferred to the caller. Otherwise, the header is deleted.

3.4.1.5.3.10.2 - CHeaderList::RemoveHeaderType Method

Removes all headers of the specified type.

C++

```
mxt_result RemoveHeaderType(IN const CString& rstrName);
```

Parameters

Parameters	Description
IN const CString& rstrName	Name of the headers to remove.

Returns

resS_OK : Headers correctly removed.

resFE_FAIL : Could not find headers of the specified type.

Description

Removes all headers of the specified type.

3.4.1.5.3.11 - CHeaderList::ReplaceHeaders Method

Appends the header list and replaces the headers that have the same type as the one in the passed list.

C++

```
void ReplaceHeaders(IN const CHeaderList& rNewHeaderList);
```

Parameters

Parameters	Description
IN const CHeaderList& rNewHeaderList	Header list to add in the list. The headers are taken from that list and appended into this list.

Description

Adds the given header list at the end of the current header list. All headers are added at the end of the list, except if other headers of the same type are present, in which case the new header replaces the headers already present.

See Also

ReplaceHeaderTypeWith (see page 164)

3.4.1.5.3.12 - CHeaderList::ReplaceHeaderTypeWith Method

Replaces all the headers of a type with a new header.

C++

```
mxt_result ReplaceHeaderTypeWith(IN TO CSipHeader* pHeaderToAdd);
```

Parameters

Parameters	Description
IN TO CSipHeader* pHeaderToAdd	Header to add to the header list. Ownership is TAKEN. It MUST NOT be NULL.

Returns

resS_OK : The header has been correctly replaced.

resFE_INVALID_ARGUMENT : pHeaderToAdd is NULL.

resFE_INVALID_ARGUMENT : pHeaderToAdd has no name.

Description

This method inserts a header in the header list after having removed all the headers of the same type.

3.4.1.5.3.13 - CHeaderList::Reset Method

Resets the header list to its initial state.

C++

```
void Reset();
```

Description

Resets the header list by deleting all headers.

3.4.1.5.3.14 - CHeaderList::Serialize Method

Outputs the headers.

C++

```
void Serialize(IN IUri::EUriType eUriType, INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
IN IUri::EUriType eUriType	Set to (eSIP eSIPS) if serializing a header within the CSipUri (see page 325) context. This affects the name and value separator, the multiheader separator, and the end of line separators. Also affects character escaping.
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the contents of the entire list into the buffer. First adds the name, then the separator, then the value of the header. Headers are comma-separated following local configuration. The header list does not add a header separator after the last header.

3.4.1.5.3.15 - CHeaderList::Sort Method

Sorts the headers using the defined header order.

C++

```
void Sort();
```

Description

This method sorts the contained headers by using the predefined header order.

See Also

The g_auHeaderOrder (see page 392) table, SortCompare.

Example

This example takes into account that the predefined header order is Via, From, To, Contact.

```
Initial list order:
Vial
To
Contact
Via2
From

After calling Sort()

Vial
Via2
From
```

To
Contact

3.4.1.5.4 - Operators

3.4.1.5.4.1 - CHeaderList::!= Operator

Inequality operator.

C++

```
bool operator !=(IN const CHeaderList& rSrc) const;
```

Parameters

Parameters	Description
IN const CHeaderList& rSrc	List against which to compare.

Returns

True if both header lists are different.

Description

Inequality operator.

See Also

operator==

3.4.1.5.4.2 - CHeaderList::= Operator

Assignment operator.

C++

```
CHeaderList& operator =(IN const CHeaderList& rSrc);
```

Parameters

Parameters	Description
IN const CHeaderList& rSrc	Header list from which to initialize.

Returns

A reference on "this" instance.

Description

Assignment operator.

3.4.1.5.4.3 - CHeaderList::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CHeaderList& rSrc) const;
```

Parameters

Parameters	Description
IN const CHeaderList& rSrc	List against which to compare.

Returns

True if both header lists are equivalent.

Description

Comparison operator. Header order is not important. All headers must be present in both lists however.

See Also

operator+=

3.4.1.6 - CHostPort Class

Class Hierarchy

`CHostPort`

C++

`class CHostPort;`

Description

This class abstracts the RFC 3261 ABNF construct "hostport".

```

RFC 3261 ABNF
hostport      = host [ ":" port ]
host          = hostname / IPv4address / IPv6reference
hostname      = *( domainlabel "." ) toplabel [ "." ]
domainlabel   = alphanum / alphanum *( alphanum / "-" ) alphanum
toplabel      = ALPHA / ALPHA *( alphanum / "-" ) alphanum
IPv4address   = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "."
IPv6reference = "[" IPv6address "]"
IPv6address   = hexpart [ ":" IPv4address ]
hexpart       = hexseq / hexseq ":" [ hexseq ] / ":" [ hexseq ]
hexseq        = hex4 *( ":" hex4 )
hex4          = 1*4HEXDIG
port          = 1*DIGIT

Corrected IPv6 ABNF (draft-gurbani-sip-ipv6-abnf-fix-00):
IPv4address   = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "."
IPv6reference = "[" IPv6address "]"
IPv6address   =
/                               6( h16 ":" ) ls32
/ [                               5( h16 ":" ) ls32
/ [ *1( h16 ":" ) h16 ] ":" 4( h16 ":" ) ls32
/ [ *2( h16 ":" ) h16 ] ":" 3( h16 ":" ) ls32
/ [ *3( h16 ":" ) h16 ] ":" 2( h16 ":" ) ls32
/ [ *4( h16 ":" ) h16 ] ":" h16 ":" ls32
/ [ *5( h16 ":" ) h16 ] ":" h16
/ [ *6( h16 ":" ) h16 ] ":" h16

h16           = 1*4HEXDIG
ls32          = ( h16 ":" h16 ) / IPv4address
IPv4address   = dec-octet "." dec-octet "." dec-octet "."
dec-octet     = DIGIT          ; 0-9
/ %x31-39 DIGIT      ; 10-99
/ "1" 2DIGIT        ; 100-199
/ "2" %x30-34 DIGIT ; 200-249
/ "25" %x30-35      ; 250-255

```

NOTES: This class must not contain the scope ID (%) when the host is an IPv6address.

Location

SipParser/CHostPort.h

Constructors

Constructor	Description
<code>CHostPort</code> (see page 168)	Constructor.

Legend

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

Destructors

Destructor	Description
<code>~CHostPort</code> (see page 169)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 175)	Inequality operator.
 = (see page 176)	Assignment operator.
 == (see page 176)	Comparison operator.

Legend

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

Methods

Method	Description
 GetHost (see page 169)	Provides access to the host.
 GetPort (see page 170)	Provides access to the port.
 Ipv6AddressToIpv6Reference (see page 170)	Converts an "Ipv6Address" to an "Ipv6Reference".
 Ipv6ReferenceToIpv6Address (see page 170)	Converts an "Ipv6Reference" to an "Ipv6Address".
 IsHostName (see page 171)	Checks if m_strHost contains a valid "hostname".
 IsIpv4Address (see page 172)	Checks if m_strHost contains a valid "Ipv4Address".
 IsIpv6Reference (see page 172)	Checks if m_strHost contains a valid "Ipv6Reference".
 Parse (see page 174)	Parses the host and optional port.
 Reset (see page 175)	Resets this hostport.
 Serialize (see page 175)	Outputs the hostport. The port is output only if different from ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168).
 SetHost (see page 175)	Sets the host.

Legend

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

3.4.1.6.1 - Data Members**3.4.1.6.1.1 - CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT Data Member**

This port value means that the default SIP port will be used.

C++

```
const uint16_t ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT;
```

Description

It also means that the SIP port will not be output in serialized packets.

3.4.1.6.2 - Constructors**3.4.1.6.2.1 - CHostPort****3.4.1.6.2.1.1 - CHostPort::CHostPort Constructor**

Constructor.

C++

```
CHostPort();
```

Description

Default constructor.

3.4.1.6.2.1.2 - CHostPort::CHostPort Constructor

Copy Constructor.

C++

```
CHostPort (IN const CHostPort& rSrc);
```

Parameters

Parameters	Description
IN const CHostPort& rSrc	Host from which to initialize.

Description

Copy constructor.

3.4.1.6.2.1.3 - CHostPort::CHostPort Constructor

Constructor.

C++

```
CHostPort (IN const CString& rstrHost, IN uint16_t uPort = ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT);
```

Parameters

Parameters	Description
IN const CString& rstrHost	Host to set.
IN uint16_t uPort = ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT	Port to set.

Description

Extended constructor.

3.4.1.6.3 - Destructors**3.4.1.6.3.1 - CHostPort::~CHostPort Destructor**

Destructor.

C++

```
virtual ~CHostPort();
```

Description

Destructor.

3.4.1.6.4 - Methods**3.4.1.6.4.1 - CHostPort::GetHost Method**

Provides access to the host.

C++

```
const CString& GetHost() const;
```

Returns

Hostname.

Description

Provides constant access to the hostname.

3.4.1.6.4.2 - GetPort**3.4.1.6.4.2.1 - CHostPort::GetPort Method**

Provides access to the port.

C++

```
uint16_t GetPort() const;
uint16_t& GetPort();
```

Returns

Port.

Description

Provides constant access to the port.

3.4.1.6.4.3 - CHostPort::Ipv6AddressToIpv6Reference Method

Converts an "Ipv6Address" to an "Ipv6Reference".

C++

```
static void Ipv6AddressToIpv6Reference( INOUT CString& rstrData );
```

Parameters

Parameters	Description
INOUT CString& rstrData	String to convert.

Description

Converts an "Ipv6Address" to an "Ipv6Reference" as per RFC 3261.

```
RFC 3261 ABNF:
IPv4address      = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference    = "[" IPv6address "]"
IPv6address      = hexpart [ ":" IPv4address ]
hexpart          = hexseq / hexseq ":" [ hexseq ] / ":" [ hexseq ]
hexseq           = hex4 *(":": hex4)
hex4             = 1*4HEXDIG
```

See Also

IsIpv6Reference (see page 172), Ipv6ReferenceToIpv6Address (see page 170)

3.4.1.6.4.4 - CHostPort::Ipv6ReferenceToIpv6Address Method

Converts an "Ipv6Reference" to an "Ipv6Address".

C++

```
static mxt_result Ipv6ReferenceToIpv6Address( INOUT CString& rstrData );
```

Parameters

Parameters	Description
INOUT CString& rstrData	String to convert.

Returns

resS_OK: The string has been converted.

resFE_INVALID_ARGUMENT: The string is not an "IPv6reference".

Description

Converts an "Ipv6Reference" to an "Ipv6Address" as per RFC 3261.

```
RFC 3261 ABNF:
IPv4address      = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference    = "[" IPv6address "]"
IPv6address      = hexpart [ ":" IPv4address ]
hexpart          = hexseq / hexseq ":" [ hexseq ] / ":" [ hexseq ]
hexseq           = hex4 *(":": hex4)
hex4             = 1*4HEXDIG
```

See Also

[IsIpv6Reference](#) (see page 172), [Ipv6ReferenceToIpv6Address](#)

3.4.1.6.4.5 - IsHostName

3.4.1.6.4.5.1 - CHostPort::IsHostName Method

Checks if m_strHost contains a valid "hostname".

C++

```
bool IsHostName() const;
```

Returns

True if the m_strHost member is a valid "hostname".

Description

Checks if m_strHost contains a valid "hostname" as per RFC 3261.

```
RFC 3261 ABNF:
hostname          = *(domainlabel ".") toplabel [ "." ]
domainlabel       = alphanum / alphanum *(alphanum / "-") alphanum
toplabel          = ALPHA / ALPHA *(alphanum / "-") alphanum
```

See Also

[IsIpv4Address](#) (see page 172), [IsIpv6Reference](#) (see page 172)

3.4.1.6.4.5.2 - CHostPort::IsHostName Method

Checks if rstrData contains a valid "hostname".

C++

```
static bool IsHostName(IN const CString& rstrData);
```

Parameters

Parameters	Description
IN const CString& rstrData	String to verify.

Returns

True if the string is a valid "hostname".

Description

Checks if the string contains a valid "hostname" as per RFC 3261.

```
RFC 3261 ABNF:
hostname          = *(domainlabel ".") toplabel [ "." ]
domainlabel       = alphanum / alphanum *(alphanum / "-") alphanum
toplabel          = ALPHA / ALPHA *(alphanum / "-") alphanum
```

See Also

[IsIpv4Address](#) (see page 172), [IsIpv6Reference](#) (see page 172)

3.4.1.6.4.6 - **IsIpv4Address**

3.4.1.6.4.6.1 - **CHostPort::IsIpv4Address** Method

Checks if m_strHost contains a valid "Ipv4Address".

C++

```
bool IsIpv4Address() const;
```

Returns

True if the m_strHost member is a valid "IPv4address".

Description

Checks if m_strHost contains a valid "IPv4address" as per RFC 3261.

RFC 3261 ABNF:	IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
----------------	---

See Also

[IsHostName](#) (see page 171), [IsIpv6Reference](#) (see page 172)

3.4.1.6.4.6.2 - **CHostPort::IsIpv4Address** Method

Checks if rstrData contains a valid "Ipv4Address".

C++

```
static bool IsIpv4Address(IN const CString& rstrData);
```

Parameters

Parameters	Description
IN const CString& rstrData	String to verify.

Returns

True if the string is a valid "IPv4address".

Description

Checks if the string contains a valid "IPv4address" as per RFC 3261.

RFC 3261 ABNF:	IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
----------------	---

See Also

[IsHostName](#) (see page 171), [IsIpv6Reference](#) (see page 172)

3.4.1.6.4.7 - **IsIpv6Reference**

3.4.1.6.4.7.1 - **CHostPort::IsIpv6Reference** Method

Checks if m_strHost contains a valid "Ipv6Reference".

C++

```
bool IsIpv6Reference() const;
```

Returns

True if the m_strHost member is a valid "IPv6reference".

Description

Checks if m_strHost contains a valid "IPv6reference" as per RFC 3261.

```

RFC 3261 ABNF:
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address = hexpart [ ":" IPv4address ]
hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq = hex4 *(":" hex4)
hex4 = 1*4HEXDIG

Corrected IPv6 ABNF (draft-gurbani-sip-ipv6-abnf-fix-00):
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address =
  / 6( h16 ":" ) ls32
  / :: 5( h16 ":" ) ls32
  / [ h16 ] :: 4( h16 ":" ) ls32
  / [ *1( h16 ":" ) h16 ] :: 3( h16 ":" ) ls32
  / [ *2( h16 ":" ) h16 ] :: 2( h16 ":" ) ls32
  / [ *3( h16 ":" ) h16 ] :: h16 ":" ls32
  / [ *4( h16 ":" ) h16 ] :: ls32
  / [ *5( h16 ":" ) h16 ] :: h16
  / [ *6( h16 ":" ) h16 ] ::

h16 = 1*4HEXDIG
ls32 = ( h16 ":" h16 ) / IPv4address
IPv4address = dec-octet "." dec-octet "." dec-octet "."
dec-octet = DIGIT ; 0-9
  / %x31-39 DIGIT ; 10-99
  / "1" 2DIGIT ; 100-199
  / "2" %x30-34 DIGIT ; 200-249
  / "25" %x30-35 ; 250-255

```

See Also

[IsHostName](#) (see page 171), [IsIpv4Address](#) (see page 172), [Ipv6ReferenceToIpv6Address](#) (see page 170), [Ipv6AddressToIpv6Reference](#) (see page 170).

3.4.1.6.4.7.2 - CHostPort::IsIpv6Reference Method

Checks if rstrData contains a valid "Ipv6reference".

C++

```
static bool IsIpv6Reference(IN const CString& rstrData);
```

Parameters

Parameters	Description
IN const CString& rstrData	String to verify.

Returns

True if the string is a valid "Ipv6reference".

Description

Checks if the string contains a valid "Ipv6reference" as per RFC 3261 with the correction brought by the draft-gurbani-sip-ipv6-abnf-fix-00. In the original ABNF, the following construct is accepted: [2001:db8::192.0.2.1]. The corrected ABNF accepts only two colons preceding the IPv4 address and rejects the constructs with three colons.

```

RFC 3261 ABNF:
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address = hexpart [ ":" IPv4address ]
hexpart = hexseq / hexseq "::" [ hexseq ] / ":" [ hexseq ]
hexseq = hex4 *(":" hex4)
hex4 = 1*4HEXDIG

Corrected IPv6 ABNF (draft-gurbani-sip-ipv6-abnf-fix-00):
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address =
  / 6( h16 ":" ) ls32
  / :: 5( h16 ":" ) ls32
  / [ h16 ] :: 4( h16 ":" ) ls32
  / [ *1( h16 ":" ) h16 ] :: 3( h16 ":" ) ls32

```

```

/ [ *2( h16 ":" ) h16 ] ":" 2( h16 ":" ) ls32
/ [ *3( h16 ":" ) h16 ] ":" h16 ":" ls32
/ [ *4( h16 ":" ) h16 ] ":" ls32
/ [ *5( h16 ":" ) h16 ] ":" h16
/ [ *6( h16 ":" ) h16 ] ":" h16

h16      = 1*4HEXDIG
ls32     = ( h16 ":" h16 ) / IPv4address
IPv4address = dec-octet "." dec-octet "." dec-octet
dec-octet = DIGIT          ; 0-9
/ %x31-39 DIGIT          ; 10-99
/ "1" 2DIGIT             ; 100-199
/ "2" %x30-34 DIGIT      ; 200-249
/ "25" %x30-35           ; 250-255

```

See Also

[IsHostName](#) (see page 171), [IsIpv4Address](#) (see page 172), [GetHexSeqLength](#), [Ipv6ReferenceToIpv6Address](#) (see page 170), [Ipv6AddressToIpv6Reference](#) (see page 170).

3.4.1.6.4.8 - CHostPort::Parse Method

Parses the host and optional port.

C++

```
mxt_result Parse(IN CStringHelper::EAllowLws eAllowLws, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN CStringHelper::EAllowLws eAllowLws	If CStringHelper::eALLOW_LWS, LWS is allowed within the host. This can only be between the hostname and port. If CStringHelper::eALLOW_LWS, trailing LWS is also skipped after successful parsing.
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. If eAllowLws is CStringHelper::eALLOW_LWS, CHostPort (see page 167) automatically advances rpcPos over any LWS encountered AFTER parsing its data. If Parse() fails, rpcPos is reset to its initial position.

Returns

resSI_SIPPARSER_DATACONT : CHostPort (see page 167) was found, optional LWS was skipped (only applies when eAllowLws is CStringHelper::eALLOW_LWS), more data follows.

resS_OK : CHostPort (see page 167) was found, optional LWS was skipped (only applies when eAllowLws is CStringHelper::eALLOW_LWS), end of data follows.

resFE_INVALID_ARGUMENT : The "hostname" part could not be validated.

resFE_INVALID_ARGUMENT : The port could not be found after the ':'.

resFE_INVALID_ARGUMENT : The port value is invalid, maybe too large.

Description

Parses a hostport ABNF construct as per RFC3261, storing the retrieved info in the local data members.

```

RFC 3261 ABNF
hostport      = host [ ":" port ]
host          = hostname / IPv4address / IPv6reference
hostname      = *(domainlabel ".") toplabel [ "." ]
domainlabel   = alphanum / alphanum *(alphanum / "-") alphanum
toplabel      = ALPHA / ALPHA *(alphanum / "-") alphanum
IPv4address   = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address   = hexpart [ ":" IPv4address ]
hexpart       = hexseq / hexseq ":" [ hexseq ] / ":" [ hexseq ]
hexseq        = hex4 *( ":" hex4)
hex4          = 1*4HEXDIG
port          = 1*DIGIT

```

3.4.1.6.4.9 - CHostPort::Reset Method

Resets this hostport.

C++

```
void Reset();
```

Description

Sets the data members to the initial state. Host is emptied and port is set to ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168).

3.4.1.6.4.10 - CHostPort::Serialize Method

Outputs the hostport. The port is output only if different from ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168).

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Where to output the internal data.

Description

Outputs the parameter's data. It outputs the hostname, and then outputs the port only if different from ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168). If the hostname is an IPv6 address, the [] are already contained in the host as the SetHost (see page 175) method has done it.

3.4.1.6.4.11 - CHostPort::SetHost Method

Sets the host.

C++

```
void SetHost(IN const CString& rstrHost);
```

Parameters

Parameters	Description
IN const CString& rstrHost	The hostname.

Description

Sets the hostname. If the specified string is an IPv6 address, the [] are added automatically. If the string is already an IPv6Reference, nothing is done and the hostname is kept as is.

See Also

GetHost (see page 169)

3.4.1.6.5 - Operators

3.4.1.6.5.1 - CHostPort::!= Operator

Inequality operator.

C++

```
bool operator !=(IN const CHostPort& rSrc) const;
```

Parameters

Parameters	Description
IN const CHostPort& rSrc	HostPort against which to compare.

Returns

True if the hostports are different.

Description

Inequality operator. Compares the hostports following RFC 3261 rules.

See Also

operator==

3.4.1.6.5.2 - =**3.4.1.6.5.2.1 - CHostPort::= Operator**

Assignment operator.

C++

```
CHostPort& operator =(IN const CHostPort& rSrc);
```

Parameters

Parameters	Description
IN const CHostPort& rSrc	Host to copy.

Returns

Returns this CHostPort (see page 167) instance.

Description

Default assignment operator.

3.4.1.6.5.2.2 - CHostPort::= Operator

Assignment operator. Assigns the host and resets the port to ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168).

C++

```
CHostPort& operator =(IN const CString& rstrHost);
```

Parameters

Parameters	Description
IN const CString& rstrHost	Host part to copy.

Returns

Returns this CHostPort (see page 167) instance.

Description

Assignment operator. Copies the host part and resets the port to ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT (see page 168).

3.4.1.6.5.3 - CHostPort::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CHostPort& rSrc) const;
```

Parameters

Parameters	Description
IN const CHostPort& rSrc	Host against which to compare.

Returns

Returns true if the host name and port are the same.

Description

Comparison operator. Compares the host and port values following RFC 3261 rules.

- RFC 3261 section 19.1.4:
- Comparison of the userinfo of SIP and SIPS URIs is **case-sensitive**. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is **case-insensitive** unless explicitly defined otherwise.
 - An IP address that is the result of a DNS lookup of a host name does not match that host name.

Implementor's note: CHostPort is considered as being part of the "all other components" mentioned above.

Section 19.1.4 Non-equivalent examples:
 sip:bob@biloxi.com (can resolve to different ports)
 sip:bob@biloxi.com:5060

A URI omitting any component with a default value will not match a URI explicitly containing that component with its default value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060.

Defining sip:user@host to not be equivalent to
 sip:user@host:5060 is a change from RFC 2543. When deriving
 addresses from URIs, equivalent addresses are expected from
 equivalent URIs. The URI sip:user@host:5060 will always
 resolve to port 5060. The URI sip:user@host may resolve to
 other ports through the DNS SRV mechanisms detailed in [4].

3.4.1.7 - CImUri Class

Class Hierarchy



C++

```
class CImUri : public CMailboxUri;
```

Description

The CImUri class is used to store, parse, and serialize IM Mailbox URIs. There are only two mailbox URI supported right now, IUri::eIM and IUri::ePRES.

An IM Mailbox URI is made up of an addr-spec, and an optional header list.

RFC 3860 IM ABNF:
 (The only difference with the normal ABNF is that the to part is mandatory)
 IM-URI = "im:" to [headers]
 to = mailbox
 headers = "?" header *("&" header)
 header = hname "=" hvalue
 hname = 1*(hnv-unreserved / unreserved / escaped)
 hvalue = *(hnv-unreserved / unreserved / escaped)
 hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "\$"

Location

SipParser/CImUri.h

See Also

CNameAddr (see page 201), CMailboxUri (see page 182)

Constructors

Constructor	Description
• ClmUri (see page 179)	Constructor.

CMailboxUri Class

CMailboxUri Class	Description
• CMailboxUri (see page 184)	Constructor.

Legend

•	Method
---	--------

Destructors

Destructor	Description
• ~ClmUri (see page 180)	Destructor.

CMailboxUri Class

CMailboxUri Class	Description
• ~CMailboxUri (see page 185)	Destructor.

IUri Class

IUri Class	Description
• ~IUri (see page 381)	Destructor.

Legend

•	Method
▼	virtual

Operators

CMailboxUri Class

CMailboxUri Class	Description
• != (see page 190)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.
• = (see page 191)	Assignment operator.
• == (see page 191)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.

Legend

•	Method
---	--------

Methods

Method	Description
• ▼ GenerateCopy (see page 180)	Generates a copy of this URI.
• ▼ GetScheme (see page 180)	Returns the URI's scheme.
• ▼ GetUriType (see page 180)	Returns the URI type.
• ▼ Parse (see page 181)	Parses a byte string into useable data.
• ▼ Serialize (see page 181)	Outputs the data member in a format that is ready to send on the network.

CMailboxUri Class

CMailboxUri Class	Description
• ▼ GenerateCopy (see page 185)	Generates a copy of this URI.
• ▼ GetDisplayName (see page 185)	
• ▼ GetHeaderList (see page 186)	Provides access to the optional header list.
• ▼ GetHostPort (see page 186)	Gets the addr-spec host contained in this URI.
• ▼ GetScheme (see page 186)	Returns the URI's scheme.

◆   GetUriType (see page 187)	Returns the URI type.
◆  GetUser (see page 332)	Gets the user if present in this URI.
◆   IsEquivalent (see page 187)	Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.
◆   Parse (see page 188)	Parses a byte string into useable data.
◆   Reset (see page 189)	Reinitializes the instance.
◆   Serialize (see page 189)	Outputs the data member in a format that is ready to send on the network.
◆  Set (see page 189)	Sets this URI with specified hostname, user, and display name.
◆  SetDisplayName (see page 190)	Sets the display name.
◆  SetHeaderList (see page 190)	Sets the header list.

IUri Class

IUri Class	Description
◆   GenerateCopy (see page 381)	Generates a copy of this URI.
◆   GetScheme (see page 382)	Returns the URI's scheme.
◆   GetUriType (see page 382)	Returns the URI type.
◆   IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
◆   Parse (see page 382)	Parses a byte string into usable data.
◆   Reset (see page 383)	Reinitializes the instance.
◆   Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

◆	Method
◆ 	virtual
◆ 	abstract

Enumerations

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.7.1 - Constructors

3.4.1.7.1.1 - CImUri

3.4.1.7.1.1.1 - CImUri::CImUri Constructor

Constructor.

C++

```
CImUri();
```

Description

Constructor.

3.4.1.7.1.1.2 - CImUri::CImUri Constructor

Copy constructor.

C++

```
CImUri(IN const CImUri& rSrc);
```

Parameters

Parameters	Description
IN const CImUri& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.7.2 - Destructors**3.4.1.7.2.1 - CImUri::~CImUri Destructor**

Destructor.

C++

```
virtual ~CImUri();
```

Description

Destructor.

3.4.1.7.3 - Methods**3.4.1.7.3.1 - CImUri::GenerateCopy Method**

Generates a copy of this URI.

C++

```
virtual GO IUUri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.7.3.2 - CImUri::GetScheme Method

Returns the URI's scheme.

C++

```
virtual const char* GetScheme() const;
```

Returns

URI Scheme.

Description

Returns the scheme.

3.4.1.7.3.3 - CImUri::GetUriType Method

Returns the URI type.

C++

```
virtual EUUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the IsEquivalent (see page 187) method.

3.4.1.7.3.4 - CImUri::Parse Method

Parses a byte string into useable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	A value of IUri::eALLOW_SPECIAL_CHARS indicates that the URI can consider the comma ',', question mark '?', or semi-colon ';' characters as part of this URI. This is as per RFC 3261 conformance item (811) (saying that URIs that contain one of these characters MUST be within angle quotes). This restriction extends to all (name-addr addr-spec) constructs (including SIP extensions). For a Mailbox family URI, this must always be allowed.
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. Initially, rpcPos points after the colon following the scheme. If Parse() fails, rpcPos points to the source of the error. URIs do not skip trailing LWS.

Returns

resSI_SIPPARSER_DATACONT : SipUri has been found, more data follows.

resS_OK : SipUri body has been found, end of data follows.

This method can also return any mxt_result that ParseDisplayName(), ParseAddrSpec() and ParseHeaders() may return.

Description

Parses an IM URI ABNF construct as per RFC 3860, storing the retrieved information in the local data members.

```
RFC 3860 IM ABNF:
(The only difference with the normal ABNF is that the to part is mandatory)
IM-URI      = "im:" to [ headers ]
to          = mailbox

headers     = "?" header *( "&" header )
header      = hname "=" hvalue
hname       = 1*( hnv-unreserved / unreserved / escaped )
hvalue      = *( hnv-unreserved / unreserved / escaped )
hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"
```

3.4.1.7.3.5 - CImUri::Serialize Method

Outputs the data member in a format that is ready to send on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the userinfo, host, and optional headers in the escaped form if necessary.

3.4.1.8 - CMailboxUri Class

Class Hierarchy



C++

```
class CMailboxUri : public IUri;
```

Description

The CMailboxUri class is used to store, parse, and serialize Mailbox URIs. The supported URI types are IUri::eMAILBOX, IUri::eIM and IUri::ePRES.

A Mailbox URI is made up of a name-addr or addr-spec, and an optional header list.

Mailbox URIs can be escaped. There are four charsets (see CToken (see page 360)) contexts associated with CMailboxUri. The user, and headers each have different lists characters that are legal and can be escaped. CMailboxUri handles all manner of escaping automatically.

```

RFC 2822 ABNF:
mailbox      = name-addr / addr-spec
name-addr    = [display-name] angle-addr
angle-addr   = [CFWS] "<" addr-spec ">" [CFWS]
display-name = phrase
addr-spec    = local-part "@" domain
local-part   = dot-atom / quoted-string
domain       = dot-atom / domain-literal
domain-literal = [CFWS] "[" *([FWS] dcontent) [FWS] "]" [CFWS]
dcontent     = dtext / quoted-pair
dtext        = NO-WS-CTL / ; Non white space controls
              %d33-90 / ; The rest of the US-ASCII
              %d94-126 ; characters not including "[",
              ; "]", or ""
dot-atom     = [CFWS] dot-atom-text [CFWS]
dot-atom-text = 1*atext *(." 1*atext)
atext        = ALPHA / DIGIT / ; Any character except controls,
              !" / "#" / ; SP, and specials.
              $" / "%" / ; Used for atoms
              "&" / ":" /
              "* " / "+" /
              "- " / "/" /
              "=" / "?" /
              "^ " / "_" /
              "` " / "{" /
              "|" / "}" /
              "~"
phrase       = 1*word
word         = atom / quoted-string
atom         = [CFWS] 1*atext [CFWS]
quoted-string = [CFWS]
              DQUOTE *([FWS] qcontent) [FWS] DQUOTE
              [CFWS]
qcontent     = qtext / quoted-pair
qtext        = NO-WS-CTL / ; Non white space controls
              %d33 / ; The rest of the US-ASCII
              %d35-91 / ; characters not including ""
              %d93-126 ; or the quote character
CFWS         = *([FWS] comment) (([FWS] comment) / FWS)
comment      = "(" *([FWS] ccontent) [FWS] ")"
ccontent     = ctext / quoted-pair / comment
ctext        = NO-WS-CTL / ; Non white space controls
              %d33-39 / ; The rest of the US-ASCII
              %d42-91 / ; characters not including "(", "
              %d93-126 ; ")", or ""
FWS          = ([*WSP CRLF] 1*WSP) / ; Folding white space
quoted-pair   = (" text)
text         = %d1-9 / ; Characters excluding CR and LF
              %d11 /
              %d12 /
  
```

%d14-127 /

Location

SipParser/CMailboxUri.h

See Also

CNameAddr (see page 201), IUri (see page 380), CHeaderList (see page 155), CSipHeader (see page 239)

Constructors

Constructor	Description
CMailboxUri (see page 184)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CMailboxUri (see page 185)	Destructor.

IUri Class

IUri Class	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 190)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.
= (see page 191)	Assignment operator.
== (see page 191)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.

Legend

	Method
--	--------

Methods

Method	Description
GenerateCopy (see page 185)	Generates a copy of this URI.
GetDisplayName (see page 185)	
GetHeaderList (see page 186)	Provides access to the optional header list.
GetHostPort (see page 186)	Gets the addr-spec host contained in this URI.
GetScheme (see page 186)	Returns the URI's scheme.
GetUriType (see page 187)	Returns the URI type.
GetUser (see page 332)	Gets the user if present in this URI.
IsEquivalent (see page 187)	Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.
Parse (see page 188)	Parses a byte string into useable data.
Reset (see page 189)	Reinitializes the instance.
Serialize (see page 189)	Outputs the data member in a format that is ready to send on the network.
Set (see page 189)	Sets this URI with specified hostname, user, and display name.
SetDisplayName (see page 190)	Sets the display name.
SetHeaderList (see page 190)	Sets the header list.

IUri Class

IUri Class	Description
GenerateCopy (see page 381)	Generates a copy of this URI.

•   GetScheme (see page 382)	Returns the URI's scheme.
•   GetUriType (see page 382)	Returns the URI type.
•   IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
•   Parse (see page 382)	Parses a byte string into usable data.
•   Reset (see page 383)	Reinitializes the instance.
•   Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	virtual
	abstract

Enumerations

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.8.1 - Constructors

3.4.1.8.1.1 - CMailboxUri

3.4.1.8.1.1.1 - CMailboxUri::CMailboxUri Constructor

Constructor.

C++

```
CMailboxUri();
```

Description

Constructor.

3.4.1.8.1.1.2 - CMailboxUri::CMailboxUri Constructor

Constructor.

C++

```
CMailboxUri(IN bool bUseDisplayName);
```

Parameters

Parameters	Description
IN bool bUseDisplayName	Tells whether or not the display name is used and displayed/serialized.

Description

Constructor.

3.4.1.8.1.1.3 - CMailboxUri::CMailboxUri Constructor

Copy constructor.

C++

```
CMailboxUri(IN const CMailboxUri& rSrc);
```

Parameters

Parameters	Description
IN const CMailboxUri& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.8.2 - Destructors**3.4.1.8.2.1 - CMailboxUri::~CMailboxUri Destructor**

Destructor.

C++

```
virtual ~CMailboxUri();
```

Description

Destructor.

3.4.1.8.3 - Methods**3.4.1.8.3.1 - CMailboxUri::GenerateCopy Method**

Generates a copy of this URI.

C++

```
virtual GO IUri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.8.3.2 - GetDisplayName**3.4.1.8.3.2.1 - CMailboxUri::GetDisplayName Method**

```
CString* GetDisplayName();
```

Returns

The display name element.

NULL if there is none.

Description

Provides access to the display name element.

3.4.1.8.3.2.2 - CMailboxUri::GetDisplayName Method

Gets the display name if present in this URI.

C++

```
const CString* GetDisplayName() const;
```

Returns

The display name element.

NULL if there is none.

Description

Provides access to the display name element.

3.4.1.8.3.3 - GetHeaderList**3.4.1.8.3.3.1 - CMailboxUri::GetHeaderList Method**

Provides access to the optional header list.

C++

```
const CHeaderList* GetHeaderList() const;
CHeaderList* GetHeaderList();
```

Returns

Pointer to the header list. It may be NULL.

Description

Provides access to the header list.

3.4.1.8.3.4 - GetHostPort**3.4.1.8.3.4.1 - CMailboxUri::GetHostPort Method**

Gets the addr-spec host contained in this URI.

C++

```
const CHostPort& GetHostPort() const;
CHostPort& GetHostPort();
```

Returns

The addr-spec host element.

Description

Provides access to the addr-spec host element.

3.4.1.8.3.5 - CMailboxUri::GetScheme Method

Returns the URI's scheme.

C++

```
virtual const char* GetScheme() const;
```

Returns

URI Scheme.

Can be NULL if the URI type is eMAILBOX.

Description

Returns the scheme.

3.4.1.8.3.6 - CMailboxUri::GetUriType Method

Returns the URI type.

C++

```
virtual EUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the IsEquivalent (see page 187) method.

3.4.1.8.3.7 - GetUser

3.4.1.8.3.7.1 - CMailboxUri:: GetUser Method

Gets the addr-spec local-part contained in this URI.

C++

```
const CString& GetUser() const;
```

Returns

The addr-spec's local-part element.

Description

Provides access to the addr-spec's local-part element.

3.4.1.8.3.8 - CMailboxUri::IsEquivalent Method

Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.

C++

```
virtual bool IsEquivalent(IN const IUri& rSrc) const;
```

Parameters

Parameters	Description
IN const IUri& rSrc	Source against which to compare.

Returns

True if the URIs are equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

Extract from RFC 3261:
19.1.4 URI Comparison

Some operations in **this** specification require determining whether two URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). Mailbox URIs are compared **for** equality according to the following rules:

- o The mailbox URIs type must first match.
- o Comparison of the mailbox URIs user is **case-sensitive**. Comparison of all other components of the URI is **case-insensitive** unless explicitly defined otherwise.

- o The ordering of the header fields is not significant in comparing mailbox URIs.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the name-addr or addr-spec must match.
 - A URI omitting the display name component will not match a URI that includes one. Thus a name-addr component cannot match an addr-spec one.
 - A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value.
- o URI header components are never ignored. Any present header component MUST be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

3.4.1.8.3.9 - CMailboxUri::Parse Method

Parses a byte string into useable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	A value of IUri::eALLOW_SPECIAL_CHARS indicates that the URI can consider the comma ',', question mark '?', or semi-colon ';' characters as part of this URI. This is as per RFC 3261 conformance item {811} (saying that URIs that contain one of these characters MUST be within angle quotes). This restriction extends to all (name-addr addr-spec) constructs (including SIP extensions). For a Mailbox family URI, this must always be allowed.
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. Initially, rpcPos points after the colon following the scheme. If Parse() fails, rpcPos points to the source of the error. URIs do not skip trailing LWS.

Returns

resSI_SIPPARSER_DATACONT : SipUri has been found, more data follows.

resS_OK : SipUri body has been found, end of data follows.

This method can also return any mxt_result that ParseDisplayName(), ParseAddrSpec() and ParseHeaders() may return.

Description

Parses a MAILBOX URI ABNF construct as per RFC 2822, storing the retrieved information in the local data members.

```
RFC 2822 ABNF:
URI          = scheme ":" [ to ] [ headers ]
to           = mailbox

mailbox      = name-addr / addr-spec
name-addr    = [display-name] angle-addr
angle-addr   = [CFWS] "<" addr-spec ">" [CFWS]
display-name = phrase
addr-spec    = local-part "@" domain
local-part   = dot-atom / quoted-string
domain       = dot-atom / domain-literal
domain-literal = [CFWS] "[" *( [FWS] dcontent ) [FWS] "]" [CFWS]
dcontent     = dtext / quoted-pair
dtext        = NO-WS-CTL / ; Non white space controls
                 %d33-90 / ; The rest of the US-ASCII
                 %d94-126 ; characters not including "[",
                 ; "]", or ""
dot-atom     = [CFWS] dot-atom-text [CFWS]
dot-atom-text = 1*atext *( ." 1*atext)
atext        = ALPHA / DIGIT / ; Any character except controls,
                 !" / "#" / ; SP, and specials.
```

```

" $" / "%" / ; Used for atoms
" &" / "' " /
" *" / "+" /
" -" / "/" /
" =" / "? " /
" ^" / "_" /
" ^" / "{" /
" |" / "}" /
" ~"

phrase      = 1*word
word        = atom / quoted-string
atom        = [CFWS] 1*atext [CFWS]
quoted-string = [CFWS]
               DQUOTE *([FWS] qcontent) [FWS] DQUOTE
               [CFWS]
qcontent    = qtext / quoted-pair
qtext       = NO-WS-CTL / ; Non white space controls
               %d33 / ; The rest of the US-ASCII
               %d35-91 / ; characters not including ""
               %d93-126 ; or the quote character

CFWS        = *([FWS] comment) ([[FWS] comment) / FWS)
comment     = "(" *([FWS] ccontent) [FWS] ")"
ccontent    = ctext / quoted-pair / comment
ctext       = NO-WS-CTL / ; Non white space controls
               %d33-39 / ; The rest of the US-ASCII
               %d42-91 / ; characters not including "(", "
               %d93-126 ; ")", or ""
FWS         = ([*WSP CRLF] 1*WSP) / ; Folding white space

headers     = "?" header *( "&" header )
header      = hname "=" hvalue
hname       = 1*( hnv-unreserved / unreserved / escaped )
hvalue      = *( hnv-unreserved / unreserved / escaped )
hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"

```

3.4.1.8.3.10 - CMailboxUri::Reset Method

Reinitializes the instance.

C++

```
virtual void Reset();
```

Description

Clears all data.

3.4.1.8.3.11 - CMailboxUri::Serialize Method

Outputs the data member in a format that is ready to send on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the userinfo, host, and optional headers in the escaped form if necessary.

3.4.1.8.3.12 - CMailboxUri::Set Method

Sets this URI with specified hostname, user, and display name.

C++

```
void Set(IN const CString& rstrHost, IN const CString& rstrUser, IN const CString* pstrDisplayName = NULL);
```

Parameters

Parameters	Description
IN const CString& rstrHost	Hostname to set.
IN const CString& rstrUser	User name.
IN const CString* pstrDisplayName = NULL	Optional display name. The default is NULL.

Description

Populates the minimal values to have a valid MailboxUri.

3.4.1.8.3.13 - CMailboxUri::SetDisplayName Method

Sets the display name.

C++

```
void SetDisplayName(IN const CString& rstrDisplayName);
```

Parameters

Parameters	Description
IN const CString& rstrDisplayName	Display name to set.

Description

Sets the display name to use.

3.4.1.8.3.14 - CMailboxUri::SetHeaderList Method

Sets the header list.

C++

```
void SetHeaderList(IN TO CHeaderList* pHeaderList);
```

Parameters

Parameters	Description
IN TO CHeaderList* pHeaderList	Header list to set. Ownership is taken. If NULL, clears the header list.

Description

Sets the header list. If a list is already present, it is released and replaced by the new one.

See Also

GetHeaderList (see page 186)

3.4.1.8.4 - Operators**3.4.1.8.4.1 - CMailboxUri::!= Operator**

Comparison operator. Compares as per RFC 3261 sect 19.1.4.

C++

```
bool operator !=(IN const CMailboxUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CMailboxUri& rSrc	Source against which to compare.

Returns

True if the URIs are not equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

Extract from RFC 3261:
19.1.4 URI Comparison

Some operations in **this** specification require determining whether two URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). Mailbox URIs are compared **for** equality according to the following rules:

- o The mailbox URIs type must first match.
- o Comparison of the display name of mailbox URIs is **case**-sensitive. Comparison of all other components of the URI is **case**-insensitive unless explicitly defined otherwise.
- o The ordering of the header fields is not significant in comparing mailbox URIs.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the name-addr or addr-spec must match.

A URI omitting the display name component will not match a URI that includes one. Thus a name-addr component cannot match an addr-spec one.

A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value.
- o URI header components are never ignored. Any present header component **MUST** be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

See Also

IsEquivalent (see page 187)

3.4.1.8.4.2 - CMailboxUri::= Operator

Assignment operator.

C++

```
CMailboxUri& operator =(IN const CMailboxUri& rSrc);
```

Parameters

Parameters	Description
IN const CMailboxUri& rSrc	Source from which to copy.

Returns

Returns a reference on "this" instance.

Description

Assignment operator.

3.4.1.8.4.3 - CMailboxUri::== Operator

Comparison operator. Compares as per RFC 3261 sect 19.1.4.

C++

```
bool operator ==(IN const CMailboxUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CMailboxUri& rSrc	Source against which to compare.

Returns

True if the URIs are equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

Extract from RFC 3261:
19.1.4 URI Comparison

Some operations in **this** specification require determining whether two URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). Mailbox URIs are compared **for** equality according to the following rules:

- o The mailbox URIs type must first match.
- o Comparison of the mailbox URIs user is **case**-sensitive.
Comparison of all other components of the URI is **case**-insensitive unless explicitly defined otherwise.
- o The ordering of the header fields is not significant in comparing mailbox URIs.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the name-addr or addr-spec must match.
A URI omitting the display name component will not match a URI that includes one. Thus a name-addr component cannot match an addr-spec one.
A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value.
- o URI header components are never ignored. Any present header component **MUST** be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

See Also

IsEquivalent (See page 187)

3.4.1.9 - CMessageSummary Class**Class Hierarchy**

CMessageSummary

C++

```
class CMessageSummary;
```

Description

The CMessageSummary class is used to store, parse, and serialize the Message waiting indicator message-summary construct. The message-summary construct is the default, text-based structure for the MWI NOTIFY body. This implementation is based on draft-ietf-sipping-mwi-04.txt.

In order for this class to parse correctly a buffer of bytes, the last byte of the buffer must be a NULL byte ('0').

```

draft-ietf-sipping-mwi-04.txt ABNF:

message-summary = msg-status-line CRLF
                  [msg-account CRLF]
                  [* (msg-summary-line CRLF)]
                  [ *opt-msg-headers ]

msg-status-line = "Messages-Waiting" HCOLON msg-status
msg-status = "yes" / "no"
msg-account = "Message-Account" HCOLON Account-URI
Account-URI = SIP-URI / SIPS-URI / absoluteURI

msg-summary-line = message-context-class HCOLON newmsgs SLASH oldmsgs
                  [ LPAREN new-urgentmsgs SLASH old-urgentmsgs RPAREN ]

newmsgs = msgcount
oldmsgs = msgcount
new-urgentmsgs = msgcount
old-urgentmsgs = msgcount
msgcount = 1*DIGIT ; MUST NOT exceed 2^32-1

RFC 3458 ABNF:
message-context-class = ( "voice-message"
                           / "fax-message"
                           / "pager-message"
                           / "multimedia-message"
                           / "text-message"
                           / "none"
)

```

Location

SipParser/CMessageSummary.h

See Also

SSummaryLine (see page 195), CHeaderList (see page 155), IUri (see page 380)

Constructors

Constructor	Description
~CMessageSummary (see page 197)	Default Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CMessageSummary (see page 197)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
= (see page 201)	Assignment Operator.

Legend

	Method
--	--------

Methods

Method	Description
GetHeaderList (see page 198)	Provides access to the optional header list.
GetMsgAccount (see page 198)	Provides access to the optional Account-URI.
GetMsgStatus (see page 198)	Returns true if the msg-status is "yes".
GetMsgSummaryLines (see page 199)	Provides access to the optional [* (msg-summary-line CRLF)].

Parse (see page 199)	Parses the data buffer, storing parsed data into the internal data members.
Reset (see page 200)	Returns all data members to their initial state.
Serialize (see page 200)	Outputs the contents of the data members into the output buffer.
SetMsgAccount (see page 200)	Sets the Account-URI.
SetMsgStatus (see page 200)	Sets the msg-status. "yes" is true, "no" is false.

Legend

	Method
--	--------

Structs

Struct	Description
SSummaryLine (see page 195)	

3.4.1.9.1 - Data Members

3.4.1.9.1.1 - CMessageSummary::ms_szMWI_MESSAGE_ACCOUNT Data Member

Start of the msg-account.

C++

```
const char* const ms_szMWI_MESSAGE_ACCOUNT;
```

3.4.1.9.1.2 - CMessageSummary::ms_szMWI_MESSAGES_WAITING Data Member

Start of the msg-status-line.

C++

```
const char* const ms_szMWI_MESSAGES_WAITING;
```

3.4.1.9.1.3 - CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_FAX Data Member

RFC3458 message-context-class value "fax-message".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_FAX;
```

3.4.1.9.1.4 - CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_MULTIMEDIA Data Member

RFC3458 message-context-class value "multimedia-message".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_MULTIMEDIA;
```

3.4.1.9.1.5 - CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_NONE Data Member

RFC3458 message-context-class value "none".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_NONE;
```

3.4.1.9.1.6 - CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_PAGER Data Member

RFC3458 message-context-class value "pager-message".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_PAGER;
```

3.4.1.9.1.7 - **CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_TEXT** Data Member

RFC3458 message-context-class value "text-message".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_TEXT;
```

3.4.1.9.1.8 - **CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_VOICE** Data Member

RFC3458 message-context-class value "voice-message".

C++

```
const char* const ms_szMWI_MSG_CONTEXT_CLASS_VOICE;
```

3.4.1.9.1.9 - **CMessageSummary::ms_szMWI_NO** Data Member

Draft-ietf-sipping-mwi-04.txt msg-status value "no".

C++

```
const char* const ms_szMWI_NO;
```

3.4.1.9.1.10 - **CMessageSummary::ms_szMWI_YES** Data Member

Draft-ietf-sipping-mwi-04.txt msg-status value "yes".

C++

```
const char* const ms_szMWI_YES;
```

3.4.1.9.1.11 - **CMessageSummary::ms_uMWI_MESSAGE_ACCOUNT_LEN_IN_BYTES** Data Member

Length of szMWI_MESSAGE_ACCOUNT.

C++

```
const uint8_t ms_uMWI_MESSAGE_ACCOUNT_LEN_IN_BYTES;
```

3.4.1.9.1.12 - **CMessageSummary::ms_uMWI_MESSAGES_WAITING_LEN_IN_BYTES** Data Member

Length of szMWI_MESSAGES_WAITING.

C++

```
const uint8_t ms_uMWI_MESSAGES_WAITING_LEN_IN_BYTES;
```

3.4.1.9.2 - Structs

3.4.1.9.2.1 - **CMessageSummary::SSummaryLine** Struct

Class Hierarchy

C++

```
struct SSummaryLine {
    CString m_strMessageContextClass;
    uint32_t m_uNewMsgs;
    uint32_t m_uOldMsgs;
    uint32_t m_uNewUrgentMsgs;
    uint32_t m_uOldUrgentMsgs;
};
```

Description

This structure contains the data to represent the msg-summary-line ABNF construct.

```
draft-ietf-sipping-mwi-04.txt ABNF:
msg-summary-line = message-context-class HCOLON newmsgs SLASH oldmsgs
[ LPAREN new-urgentmsgs SLASH old-urgentmsgs RPAREN ]

newmsgs = msgcount
oldmsgs = msgcount
new-urgentmsgs = msgcount
old-urgentmsgs = msgcount
msgcount = 1*DIGIT ; MUST NOT exceed 2^32-1

RFC 3458 ABNF:
message-context-class = ( "voice-message"
/ "fax-message"
/ "pager-message"
/ "multimedia-message"
/ "text-message"
/ "none"
)
```

Location

SipParser/CMessageSummary.h

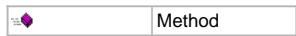
See Also

CMessageSummary (see page 192)

Constructors

Constructor	Description
SSummaryLine (see page 197)	Constructor, initializes all numbers to 0.

Legend



3.4.1.9.2.1.1 - Data Members

3.4.1.9.2.1.1.1 - CMessageSummary::SSummaryLine::m_strMessageContextClass Data Member

Contains the message-context-class construct.

C++

```
CString m_strMessageContextClass;
```

3.4.1.9.2.1.1.2 - CMessageSummary::SSummaryLine::m_uNewMsgs Data Member

Contains the newmsgs construct.

C++

```
uint32_t m_uNewMsgs;
```

3.4.1.9.2.1.1.3 - CMessageSummary::SSummaryLine::m_uNewUrgentMsgs Data Member

Contains the new-urgentmsgs construct.

C++

```
uint32_t m_uNewUrgentMsgs;
```

3.4.1.9.2.1.1.4 - CMessageSummary::SSummaryLine::m_uOldMsgs Data Member

Contains the oldmsgs construct.

C++

```
uint32_t m_uOldMsgs;
```

3.4.1.9.2.1.1.5 - CMessageSummary::SSummaryLine::m_uOldUrgentMsgs Data Member

Contains the old-urgentmsgs construct.

C++

```
uint32_t m_uOldUrgentMsgs;
```

3.4.1.9.2.1.2 - Constructors**3.4.1.9.2.1.2.1 - CMessageSummary::SSummaryLine::SSummaryLine Constructor**

Constructor, initializes all numbers to 0.

C++

```
SSummaryLine();
```

3.4.1.9.3 - Constructors**3.4.1.9.3.1 - CMessageSummary****3.4.1.9.3.1.1 - CMessageSummary::CMessageSummary Constructor**

Default Constructor.

C++

```
CMessageSummary();
```

Description

Constructor.

3.4.1.9.3.1.2 - CMessageSummary::CMessageSummary Constructor

Copy Constructor.

C++

```
CMessageSummary( IN const CMessageSummary& rSrc );
```

Parameters

Parameters	Description
IN const CMessageSummary& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.9.4 - Destructors**3.4.1.9.4.1 - CMessageSummary::~CMessageSummary Destructor**

Destructor.

C++

```
virtual ~CMessageSummary();
```

Description

Destructor.

3.4.1.9.5 - Methods**3.4.1.9.5.1 - GetHeaderList****3.4.1.9.5.1.1 - CMessageSummary::GetHeaderList Method**

Provides access to the optional header list.

C++

```
const CHeaderList& GetHeaderList() const;  
CHeaderList& GetHeaderList();
```

Returns

Constant reference to the list of headers.

Description

Provides access to the list of headers.

3.4.1.9.5.2 - GetMsgAccount**3.4.1.9.5.2.1 - CMessageSummary::GetMsgAccount Method**

Provides access to the optional Account-URI.

C++

```
const IUri* GetMsgAccount() const;  
IUri* GetMsgAccount();
```

Returns

Pointer to the Account-URI. It may be NULL.

Description

Provides access to the optional Account-URI.

See Also

[SetMsgAccount](#) (see page 200)

3.4.1.9.5.3 - CMessageSummary::GetMsgStatus Method

Returns true if the msg-status is "yes".

C++

```
bool GetMsgStatus() const;
```

Returns

True if the msg-status is "yes", false otherwise.

Description

Provides access to the msg-status value. True means that the set or parsed msg-status is "yes". Any other msg-status is processed as "no".

See Also

[SetMsgStatus](#) (see page 200)

3.4.1.9.5.4 - GetMsgSummaryLines

3.4.1.9.5.4.1 - CMessageSummary::GetMsgSummaryLines Method

Provides access to the optional [*(msg-summary-line CRLF)].

C++

```
const CVList<SSummaryLine>& GetMsgSummaryLines() const;
CVList<SSummaryLine>& GetMsgSummaryLines();
```

Returns

Constant reference to the list of summary lines.

Description

Provides access to the list of summary lines.

3.4.1.9.5.5 - CMessageSummary::Parse Method

Parses the data buffer, storing parsed data into the internal data members.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to a buffer containing the text version of the MWI-NOTIFY's body. This method parses the message-summary construct as per draft-ietf-sipping-mwi-04.txt ABNF. rpcPos is adjusted as parsing progresses. If this method fails, rpcPos points to the start of the data construct where the error is detected. If the method succeeds, rpcPos points to after the last CRLF, probably to the NULL terminator.

Returns

resSI_SIPPARSER_DATACONT : Success, more data follows.

resS_OK : Success, end of data follows.

resFE_INVALID_ARGUMENT : Failure, the "msg-status-line CRLF" is invalid.

resFE_INVALID_ARGUMENT : Failure, the [msg-account CRLF] is invalid.

resFE_INVALID_ARGUMENT : Failure, the [*(msg-summary-line CRLF)] is invalid.

resFE_INVALID_ARGUMENT : Failure, the [*opt-msg-headers] is invalid.

Description

Parses the messsage-summary construct and stores the data into data members. If the method fails, the object is reset.

```
draft-ietf-sipping-mwi-04.txt ABNF:

messsage-summary = msg-status-line CRLF
                  [msg-account CRLF]
                  [*(msg-summary-line CRLF)]
                  [ *opt-msg-headers ]

msg-status-line = "Messages-Waiting" HCOLON msg-status
msg-status = "yes" / "no"
msg-account = "Message-Account" HCOLON Account-URI
Account-URI = SIP-URI / SIPS-URI / absoluteURI

msg-summary-line = message-context-class HCOLON newmsgs SLASH oldmsgs
                  [ LPAREN new-urgentmsgs SLASH old-urgentmsgs RPAREN ]

newmsgs = msgcount
oldmsgs = msgcount
new-urgentmsgs = msgcount
```

```

old-urgentmsgs = msgcount
msgcount = 1*DIGIT ; MUST NOT exceed 2^32-1

RFC 3458 ABNF:
message-context-class = ( "voice-message"
/ "fax-message"
/ "pager-message"
/ "multimedia-message"
/ "text-message"
/ "none"
)

```

3.4.1.9.5.6 - CMessageSummary::Reset Method

Returns all data members to their initial state.

C++

```
void Reset();
```

Description

Returns all data members to their initial state.

3.4.1.9.5.7 - CMessageSummary::Serialize Method

Outputs the contents of the data members into the output buffer.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the status-line, optional Account-URI, optional summary lines, and optional headers. Note that the headers cannot be separated into multiple CRLF-separated blocks, they are all output as a single block.

3.4.1.9.5.8 - CMessageSummary::SetMsgAccount Method

Sets the Account-URI.

C++

```
void SetMsgAccount(IN TO IUri* pAccountUri);
```

Parameters

Parameters	Description
IN TO IUri* pAccountUri	Pointer to the Account-URI to set. Ownership is taken.

Description

Sets the optional Account-URI. If another Account-URI is already present, it is released, and the new Account-URI replaces it.

See Also

GetMsgAccount (see page 198)

3.4.1.9.5.9 - CMessageSummary::SetMsgStatus Method

Sets the msg-status. "yes" is true, "no" is false.

C++

```
void SetMsgStatus(IN bool bMessages);
```

Parameters

Parameters	Description
IN bool bMessages	The msg-status. True means that messages are present.

Description

Sets the msg-status value.

See Also

GetMsgStatus (see page 198)

3.4.1.9.6 - Operators

3.4.1.9.6.1 - CMessageSummary::= Operator

Assignment Operator.

C++

```
CMessageSummary& operator =(IN const CMessageSummary& rSrc);
```

Parameters

Parameters	Description
IN const CMessageSummary& rSrc	Source from which to copy.

Returns

Reference on "this" instance.

Description

Assignment operator.

3.4.1.10 - CNameAddr Class

Class Hierarchy

```
CNameAddr
```

C++

```
class CNameAddr;
```

Description

The CNameAddr class handles the storage, parsing, and serialization of the (name-addr | addr-spec) construct. It is a URI with optional display name and angle quotes. The angle quotes may be mandatory through the use of the bForceNameAddr parameter at parse time. The angle quotes are always output at serialization time.

The CNameAddr class is a simple container comprised of a token and an URI. Refer to the aggregate class's documentation for further details.

The display name is handled in a very simplified way. Since the stack does not care what the display-name is, it outputs it directly. The user has the responsibility of following RFC 3261 rules regarding the display-name, such as having the enclosing quotes when the display-name is a quoted string.

```
RFC 3261 ABNF (when forcing the name-addr criterion,
  set eAngleBracket to eMANDATORY_ANGLE_BRACKET) :
  name-addr  = [ display-name ] "<" addr-spec ">"
  display-name = *token | quoted-string
  addr-spec   = SIP-URI / SIPS-URI / absoluteURI

RFC 3261 extended ABNF (when relaxing the name-addr criterion,
  set eAngleBracket to eOPTIONAL_ANGLE_BRACKET) :
  name-addr  = ([ display-name ] "<" addr-spec ">") / addr-spec
  display-name = *token | quoted-string
  addr-spec   = SIP-URI / SIPS-URI / absoluteURI
```

Note that the extended ABNF is more flexible than the "official" RFC 3261 ABNF.

Location

SipParser/CNameAddr.h

See Also

CToken (see page 360), IUri (see page 380), CSipUri (see page 325), CAbsoluteUri (see page 128), CImUri (see page 177), CPresUri (see page 211)

Constructors

Constructor	Description
~CNameAddr (see page 203)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CNameAddr (see page 203)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 208)	Comparison operator. Compares only the URI.
= (see page 209)	Assignment operator.
== (see page 209)	Comparison operator. Compares only the URI.

Legend

	Method
--	--------

Methods

Method	Description
GetDisplayName (see page 204)	Provides access to the display name.
GetMailboxUri (see page 204)	Provides access to the addr-spec and casts it to CMailBoxUri if applicable. It can then be recast as IM or PRES URIs.
GetSipUri (see page 204)	Provides access to the addr-spec and casts it to CSipUri (see page 325) if applicable.
GetUri (see page 205)	Provides access to the addr-spec.
Parse (see page 205)	Parses the optional display name and following URI.
Reset (see page 206)	Resets data members to initial state.
Serialize (see page 206)	Outputs the optional display name and following URI.
SetMailboxUri (see page 207)	Sets the URI to a pres or im URI.
SetSipUri (see page 207)	Sets the URI to a SIPURI.
SetUri (see page 207)	Sets the URI value for the addr-spec part.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
EAngleBracket (see page 210)	

3.4.1.10.1 - Constructors

3.4.1.10.1.1 - CNameAddr

3.4.1.10.1.1.1 - CNameAddr::CNameAddr Constructor

Constructor.

C++

```
CNameAddr();
```

Description

Constructor.

3.4.1.10.1.1.2 - CNameAddr::CNameAddr Constructor

Copy constructor.

C++

```
CNameAddr(IN const CNameAddr& rSrc);
```

Parameters

Parameters	Description
IN const CNameAddr& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.10.1.1.3 - CNameAddr::CNameAddr Constructor

Extended constructor. Assumes an empty display name.

C++

```
CNameAddr(IN const IUri& rSrc);
```

Parameters

Parameters	Description
IN const IUri& rSrc	Source from which to copy.

Description

Extended constructor. Initializes the display-name to an empty value.

3.4.1.10.2 - Destructors

3.4.1.10.2.1 - CNameAddr::~CNameAddr Destructor

Destructor.

C++

```
virtual ~CNameAddr();
```

Description

Destructor.

3.4.1.10.3 - Methods

3.4.1.10.3.1 - GetDisplayName

3.4.1.10.3.1.1 - CNameAddr::GetDisplayName Method

Provides access to the display name.

C++

```
const CToken& GetDisplayName() const;
CToken& GetDisplayName();
```

Returns

Constant reference on the display name. If the display name is a quoted string, the quotes are included in the stored data.

Description

Provides access to the display name.

3.4.1.10.3.2 - GetMailboxUri

3.4.1.10.3.2.1 - CNameAddr::GetMailboxUri Method

Provides access to the addr-spec and casts it to CMailBoxUri if applicable. It can then be recast as IM or PRES URIs.

C++

```
CMailboxUri* GetMailboxUri();
```

Returns

Pointer to the addr-spec, if it is a mailbox URI.

NULL if the URI is not a mailbox URI or if there is no URI.

Description

Provides access to the URI in mailbox form.

See Also

[SetUri](#) (see page 207), [GetUri](#) (see page 205), [SetMailboxUri](#) (see page 207)

3.4.1.10.3.2.2 - CNameAddr::GetMailboxUri Method

Provides access to the addr-spec and casts it to CMailboxUri (see page 182) if applicable. Depending on the type, it may then be recasted as IM or PRES URIs.

C++

```
const CMailboxUri* GetMailboxUri() const;
```

Returns

Pointer to the addr-spec, if it is a mailbox URI.

NULL if the URI is not a mailbox URI or if there is no URI.

Description

Provides access to the URI in mailbox form.

See Also

[SetUri](#) (see page 207), [GetUri](#) (see page 205), [SetMailboxUri](#) (see page 207)

3.4.1.10.3.3 - GetSipUri

3.4.1.10.3.3.1 - CNameAddr::GetSipUri Method

Provides access to the addr-spec and casts it to CSipUri (see page 325) if applicable.

C++

```
inline CSipUri* GetSipUri();
```

Returns

Pointer to the addr-spec, if it is a SipUri. It can be NULL if the URI is not SIP or if the URI is simply absent.

Description

Provides access to the URI in SIP form.

See Also

[SetUri](#) (see page 207), [GetUri](#) (see page 205), [SetSipUri](#) (see page 207)

3.4.1.10.3.3.2 - CNameAddr::GetSipUri Method

Provides access to the addr-spec and casts it to CSipUri (see page 325) if applicable.

C++

```
inline const CSipUri* GetSipUri() const;
```

Returns

Pointer to the addr-spec, if it is a SipUri. It can be NULL if the URI is not SIP or if the URI is simply absent.

Description

Provides access to the URI in SIP form.

See Also

[SetUri](#) (see page 207), [GetUri](#) (see page 205), [SetSipUri](#) (see page 207)

3.4.1.10.3.4 - GetUri**3.4.1.10.3.4.1 - CNameAddr::GetUri Method**

Provides access to the addr-spec.

C++

```
const IUri* GetUri() const;
IUri* GetUri();
```

Returns

Pointer to the addr-spec. It may be NULL.

Description

Provides access to the addr-spec URI.

See Also

[SetUri](#) (see page 207), [GetSipUri](#) (see page 204), [SetSipUri](#) (see page 207)

3.4.1.10.3.5 - CNameAddr::Parse Method

Parses the optional display name and following URI.

C++

```
mxt_result Parse(INOUT const char*& rpcPos, IN EAngleBracket eAngleBracket = eOPTIONAL_ANGLE_BRACKET);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. This method also skips trailing LWS after the name-addr has been parsed.
eAngleQuote	Sets the parsing criterion for the name-addr structure. In essence, the difference is that when this parameter is set to eMANDATORY_ANGLE_BRACKET, the angle brackets become mandatory.

Returns

resSI_SIPPARSER_DATACONT : Parsed the name-addr, skipped LWS, more data follows.

resS_OK : Parsed the name-addr, skipped LWS, end of data follows.

resFE_UNEXPECTED : The name-addr has been forced and angle brackets have not been found.

resFE_UNEXPECTED : A matching closing angle bracket has not been found.

This method can also return any error mxt_result that ParseDisplayName() and CUriFactory::ParseUri (see page 379)() can return.

Description

Parses the optional display-name, angle brackets, and URI. The URI part is parsed using the CUriFactory (see page 378). The display-name is stored in a CToken (see page 360). If the parsed display-name is a quoted-string, the enclosing quotes are also stored in the CToken (see page 360).

```

RFC 3261 ABNF (when eAngleBracket is eMANDATORY_ANGLE_BRACKET) :
  name-addr      = [ display-name ] "<" addr-spec ">"
  display-name   = *token | quoted-string
  addr-spec      = SIP-URI / SIPS-URI / absoluteURI

RFC 3261 extended ABNF (when eAngleBracket is eOPTIONAL_ANGLE_BRACKET) :
  name-addr      = ([ display-name ] "<" addr-spec ">") / addr-spec
  display-name   = *token | quoted-string
  addr-spec      = SIP-URI / SIPS-URI / absoluteURI

Note that the extended ABNF is more flexible than the "official"
RFC 3261 ABNF.

```

3.4.1.10.3.6 - CNameAddr::Reset Method

Resets data members to initial state.

C++

```
void Reset();
```

Description

Releases the URI and clears the display-name.

3.4.1.10.3.7 - CNameAddr::Serialize Method

Outputs the optional display name and following URI.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the object's data into the buffer. Outputs the display-name and addr-spec. If the display-name is non-empty, the bracket enclosing is automatically forced.

3.4.1.10.3.8 - CNameAddr::SetMailboxUri Method

Sets the URI to a pres or im URI.

C++

```
void SetMailboxUri(IN IUri::EUriType eType, IN const CString& rstrUser, IN const CString& rstrHost, IN const
CString& rstrDisplayName = CString());
```

Parameters

Parameters	Description
IN IUri::EUriType eType	Type of MAILBOX URI to set.
IN const CString& rstrUser	User name to set.
IN const CString& rstrHost	Hostname to set.
IN const CString& rstrDisplayName = CString()	Display-name to set. Default is empty.

Description

Sets the addr-spec part as a mailbox URI. If a URI is already configured, it is released and replaced by the new URI.

The display name will be used for the name addr.

See Also

GetMailboxUri (see page 204), SetUri (see page 207), GetUri (see page 205)

3.4.1.10.3.9 - CNameAddr::SetSipUri Method

Sets the URI to a SIPURI.

C++

```
void SetSipUri(IN const CString& rstrHost, IN uint16_t uPort = CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT,
IN const CString& rstrUser = CString(), IN CSipUri::ESecurityFlag eSecured = CSipUri::eUNSECURE, IN const
CString& rstrDisplayName = CString());
```

Parameters

Parameters	Description
IN const CString& rstrHost	Hostname to set.
IN uint16_t uPort = CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT	Port to set.
IN const CString& rstrUser = CString()	Optional user name.
IN CSipUri::ESecurityFlag eSecured = CSipUri::eUNSECURE	Security setting, eSECURE means SIPS.
IN const CString& rstrDisplayName = CString()	Display-name to set. It may be empty.

Description

Sets the addr-spec part as a SipUri. If a URI is already configured, it is released and replaced by the new URI.

See Also

GetSipUri (see page 204), SetUri (see page 207), GetUri (see page 205)

3.4.1.10.3.10 - CNameAddr::SetUri Method

Sets the URI value for the addr-spec part.

C++

```
void SetUri(IN TO IUri* pUri, IN const CString& rstrDisplayName = CString());
```

Parameters

Parameters	Description
IN TO IUri* pUri	URI to set. Ownership is taken.
IN const CString& rstrDisplayName = CString()	Optional display-name.

Description

Sets the addr-spec and the optional display name.

See Also

GetUri (see page 205), SetSipUri (see page 207), GetSipUri (see page 204)

3.4.1.10.4 - Operators

3.4.1.10.4.1 - !=

3.4.1.10.4.1.1 - CNameAddr::!= Operator

Comparison operator. Compares only the URI.

C++

```
bool operator !=(IN const CNameAddr& rSrc) const;
```

Parameters

Parameters	Description
IN const CNameAddr& rSrc	Source against which to compare.

Returns

True if the addr-spec parts are not equivalent.

Description

Comparison operator. Compares the URIs only, as per RFC 3261 section 20.20:

"Two From header fields are equivalent if their URIs match, and their parameters match. Extension parameters in one header field, not present in the other are ignored for the purposes of comparison. This means that the display name and presence or absence of angle brackets do not affect matching."

Notes

If the source m_pAddrSpec and the local m_pAddrSpec are NULL, the comparison returns true. This is because while this object is in its initial state, it is considered invalid and therefore the comparison makes no sense. Consequently the comparison operator should be avoided on invalid CNameAddr (see page 201).

3.4.1.10.4.1.2 - CNameAddr::!= Operator

Comparison operator. Compares only the URI.

C++

```
bool operator !=(IN const IUri* pSrc) const;
```

Parameters

Parameters	Description
IN const IUri* pSrc	Source against which to compare.

Returns

True if the addr-spec parts are not equivalent.

Description

Comparison operator. Compares the URIs only, as per RFC 3261 section 20.20:

"Two From header fields are equivalent if their URIs match, and their parameters match. Extension parameters in one header field, not present in the other are ignored for the purposes of comparison. This means that the display name and presence or absence of angle brackets

```
do not affect matching."
```

Notes

If the source `m_pAddrSpec` and the provided `lUri` (see page 380) are `NULL`, the comparison returns true. This is because while this object is in its initial state, it is considered invalid and therefore the comparison makes no sense. Consequently the comparison operator should be avoided on invalid `CNameAddr` (see page 201).

3.4.1.10.4.2 - =

3.4.1.10.4.2.1 - `CNameAddr::= Operator`

Assignment operator.

C++

```
CNameAddr& operator =(IN const CNameAddr& rSrc);
```

Parameters

Parameters	Description
IN const CNameAddr& rSrc	Source from which to copy.

Returns

A reference on "this" instance.

Description

Assignment operator.

See Also

`operator=@IN const lUri` (see page 380)& `rSrc`

3.4.1.10.4.2.2 - `CNameAddr::= Operator`

Assignment operator. Assumes an empty display name.

C++

```
CNameAddr& operator =(IN const lUri& rSrc);
```

Parameters

Parameters	Description
IN const lUri& rSrc	Source from which to copy.

Returns

A reference on "this" instance.

Description

Assignment operator. Copies the URI and resets the display-name.

See Also

`operator=@IN const CNameAddr` (see page 201)& `rSrc`

3.4.1.10.4.3 - ==

3.4.1.10.4.3.1 - `CNameAddr::== Operator`

Comparison operator. Compares only the URI.

C++

```
bool operator ==(IN const CNameAddr& rSrc) const;
```

Parameters

Parameters	Description
IN const CNameAddr& rSrc	Source against which to compare.

Returns

True if the addr-spec parts are equivalent.

Description

Comparison operator. Compares the URIs only, as per RFC 3261 section 20.20:

"Two From header fields are equivalent if their URIs match, and their parameters match. Extension parameters in one header field, not present in the other are ignored for the purposes of comparison. This means that the display name and presence or absence of angle brackets do not affect matching."

Notes

If the source m_pAddrSpec and the local m_pAddrSpec are NULL, the comparison returns false. This is because while this object is in its initial state, it is considered invalid and therefore the comparison makes no sense. Consequently the comparison operator should be avoided on invalid CNameAddr (see page 201).

3.4.1.10.4.3.2 - CNameAddr::== Operator

Comparison operator. Compares only the URI.

C++

```
bool operator ==(IN const IUuri* pSrc) const;
```

Parameters

Parameters	Description
IN const IUuri* pSrc	Source against which to compare. NULL pointers are not equivalent.

Returns

True if the addr-spec parts are equivalent.

Description

Comparison operator. Compares the URIs only, as per RFC 3261 section 20.20:

"Two From header fields are equivalent if their URIs match, and their parameters match. Extension parameters in one header field, not present in the other are ignored for the purposes of comparison. This means that the display name and presence or absence of angle brackets do not affect matching."

Notes

If the source m_pAddrSpec and the provided IUuri (see page 380) are NULL, the comparison returns false. This is because while this object is in its initial state, it is considered invalid and therefore the comparison makes no sense. Consequently the comparison operator should be avoided on invalid CNameAddr (see page 201).

3.4.1.10.5 - Enumerations

3.4.1.10.5.1 - CNameAddr::EAngleBracket Enumeration

```
enum EAngleBracket {
    eMANDATORY_ANGLE_BRACKET,
    eOPTIONAL_ANGLE_BRACKET
};
```

Description

Indicates whether or not angle bracket are mandatory.

Members

Members	Description
eMANDATORY_ANGLE_BRACKET	Angle brackets must be present when parsing the string.
eOPTIONAL_ANGLE_BRACKET	Angle brackets must optionally be present when parsing the string.

3.4.1.11 - CPresUri Class

Class Hierarchy



C++

```
class CPresUri : public CMailboxUri;
```

Description

The CPresUri class is used to store, parse, and serialize PRES Mailbox URIs. There are only two mailbox URI supported right now, IUri::eIM and IUri::ePRES.

A PRES Mailbox URI is made up of an addr-spec, and an optional header list.

```

RFC 3859 PRES ABNF:
(The only difference with the normal ABNF is that the to part is mandatory)
PRES-URI      = "pres:" to [ headers ]
to            = mailbox

headers      = "?" header *( "&" header )
header       = hname "=" hvalue
hname        = 1*( hnv-unreserved / unreserved / escaped )
hvalue       = *( hnv-unreserved / unreserved / escaped )
hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"

```

Location

SipParser/CPresUri.h

See Also

CNameAddr (see page 201), CMailboxUri (see page 182)

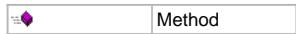
Constructors

Constructor	Description
CPresUri (see page 213)	Constructor.

CMailboxUri Class

CMailboxUri Class	Description
CMailboxUri (see page 184)	Constructor.

Legend



Destructors

Destructor	Description
~CPresUri (see page 213)	Destructor.

CMailboxUri Class

CMailboxUri Class	Description
~CMailboxUri (see page 185)	Destructor.

IUri Class

IUri Class	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Operators

CMailboxUri Class

CMailboxUri Class	Description
 != (See page 190)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.
 = (See page 191)	Assignment operator.
 == (See page 191)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.

Legend

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

Methods

Method	Description
  GenerateCopy (See page 213)	Generates a copy of this URI.
  GetScheme (See page 214)	Returns the URI's scheme.
  GetUriType (See page 214)	Returns the URI type.
  Parse (See page 214)	Parses a byte string into useable data.
  Serialize (See page 214)	Outputs the data member in a format that is ready to send on the network.

CMailboxUri Class

CMailboxUri Class	Description
  GenerateCopy (See page 185)	Generates a copy of this URI.
  GetDisplayName (See page 185)	
  GetHeaderList (See page 186)	Provides access to the optional header list.
  GetHostPort (See page 186)	Gets the addr-spec host contained in this URI.
  GetScheme (See page 186)	Returns the URI's scheme.
  GetUriType (See page 187)	Returns the URI type.
  GetUser (See page 332)	Gets the user if present in this URI.
  IsEquivalent (See page 187)	Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.
  Parse (See page 188)	Parses a byte string into useable data.
  Reset (See page 189)	Reinitializes the instance.
  Serialize (See page 189)	Outputs the data member in a format that is ready to send on the network.
  Set (See page 189)	Sets this URI with specified hostname, user, and display name.
  SetDisplayName (See page 190)	Sets the display name.
  SetHeaderList (See page 190)	Sets the header list.

IUri Class

IUri Class	Description
  GenerateCopy (See page 381)	Generates a copy of this URI.
  GetScheme (See page 382)	Returns the URI's scheme.
  GetUriType (See page 382)	Returns the URI type.
  IsEquivalent (See page 382)	Compares the given URI with this instance by using applicable RFC rules.
  Parse (See page 382)	Parses a byte string into usable data.
  Reset (See page 383)	Reinitializes the instance.
  Serialize (See page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	virtual
	abstract

Enumerations

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.11.1 - Constructors

3.4.1.11.1.1 - CPresUri

3.4.1.11.1.1.1 - CPresUri::CPresUri Constructor

Constructor.

C++

```
CPresUri();
```

Description

Constructor.

3.4.1.11.1.1.2 - CPresUri::CPresUri Constructor

Copy constructor.

C++

```
CPresUri(IN const CPresUri& rSrc);
```

Parameters

Parameters	Description
IN const CPresUri& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.11.2 - Destructors

3.4.1.11.2.1 - CPresUri::~CPresUri Destructor

Destructor.

C++

```
virtual ~CPresUri();
```

Description

Destructor.

3.4.1.11.3 - Methods

3.4.1.11.3.1 - CPresUri::GenerateCopy Method

Generates a copy of this URI.

C++

```
virtual GO IUUri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.11.3.2 - CPresUri::GetScheme Method

Returns the URI's scheme.

C++

```
virtual const char* GetScheme() const;
```

Returns

URI Scheme.

Description

Returns the scheme.

3.4.1.11.3.3 - CPresUri::GetUriType Method

Returns the URI type.

C++

```
virtual EUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the IsEquivalent (see page 187) method.

3.4.1.11.3.4 - CPresUri::Parse Method

Parses a byte string into useable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

3.4.1.11.3.5 - CPresUri::Serialize Method

Outputs the data member in a format that is ready to send on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the userinfo, host, and optional headers in the escaped form if necessary.

3.4.1.12 - CQuotedString Class

Class Hierarchy

CQuotedString

C++

```
class CQuotedString;
```

Description

This object abstracts a quoted string. The parsing mechanism is very much simplified. This object does not do any character escaping or UTF8 conversion. It only finds the matching ending quote. Users of this class can set a string that omits the quotes since the Serialization process adds them automatically. When parsing, the stored data does not include the enclosing quotes.

Location

SipParser/CQuotedString.h

Constructors

Constructor	Description
• CQuotedString (see page 216)	Constructor.

Legend

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

Destructors

Destructor	Description
• ~CQuotedString (see page 216)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
• = (see page 218)	Assignment operators.

Legend

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

Methods

Method	Description
• GetString (see page 217)	Provides access to the internal string. The set data should omit the quotes since they are automatically added when Serializing. Otherwise, another set of quotes are added.
• Parse (see page 217)	Parses a quoted string structure. Simply finds the matching quote.
• Reset (see page 217)	Clears the quoted string.
• Serialize (see page 217)	Outputs the quoted string to the blob. The output always contains quotes.

Legend

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

3.4.1.12.1 - Constructors

3.4.1.12.1.1 - CQuotedString

3.4.1.12.1.1.1 - CQuotedString::CQuotedString Constructor

Constructor.

C++

```
CQuotedString();
```

Description

Default constructor.

3.4.1.12.1.1.2 - CQuotedString::CQuotedString Constructor

Constructor.

C++

```
CQuotedString(IN const CQuotedString& rSrc);
```

Parameters

Parameters	Description
IN const CQuotedString& rSrc	Quoted string to copy.

Description

Copy constructor.

3.4.1.12.1.1.3 - CQuotedString::CQuotedString Constructor

Constructor.

C++

```
CQuotedString(IN const CString& rstr);
```

Parameters

Parameters	Description
IN const CString& rstr	Data.

Description

Extended constructor.

3.4.1.12.2 - Destructors

3.4.1.12.2.1 - CQuotedString::~CQuotedString Destructor

Destructor.

C++

```
virtual ~CQuotedString();
```

Description

Destructor.

3.4.1.12.3 - Methods

3.4.1.12.3.1 - GetString

3.4.1.12.3.1.1 - CQuotedString::GetString Method

Provides access to the internal string. The set data should omit the quotes since they are automatically added when Serializing. Otherwise, another set of quotes are added.

C++

```
const CString& GetString() const;
CString& GetString();
```

Returns

The contained quoted string.

Description

Provides access to the quoted string data.

3.4.1.12.3.2 - CQuotedString::Parse Method

Parses a quoted string structure. Simply finds the matching quote.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. CQuotedString (see page 215) automatically advances rpcPos over any LWS encountered AFTER parsing its data. If Parse() fails, rpcPos is left untouched.

Returns

resSI_SIPPARSER_DATACONT : Found a quoted string, skipped optional LWS, more data follows.

resS_OK : Found a quoted string, skipped optional LWS, end of data follows.

resFE_UNEXPECTED : Initially, rpcPos did not point to the starting quote, processing ended there. LWS was not skipped.

resFE_UNEXPECTED : Did not find the ending quotes, processing ended there. LWS was not skipped.

Description

Parses a quoted string, advancing rpcPos as it goes. Processing continues until the matching closing quote is found. No unescaping or UTF8 conversion is made, and no validation of the quoted string's validity is made whatsoever. The stored data does not include the starting and ending quotes.

3.4.1.12.3.3 - CQuotedString::Reset Method

Clears the quoted string.

C++

```
void Reset();
```

Description

Clears the internal data (the quoted string).

3.4.1.12.3.4 - CQuotedString::Serialize Method

Outputs the quoted string to the blob. The output always contains quotes.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT Cblob& rBlob	Where to output the internal data.

Description

Outputs the CQuotedString (see page 215)'s data. Since the internal data does not contain starting and ending quotes, these are added automatically. The output data always contains quotes.

3.4.1.12.4 - Operators

3.4.1.12.4.1 - =

3.4.1.12.4.1.1 - CQuotedString::= Operator

Assignment operators.

C++

```
CQuotedString& operator =(IN const CQuotedString& rSrc);
CQuotedString& operator =(IN const CString& rstrSrc);
```

Parameters

Parameters	Description
IN const CQuotedString& rSrc	String to copy.
IN const CQuotedString& rSrc	Quoted string to copy.

Returns

This method returns the CQuotedString (see page 215) instance.

Description

Default assignment operator.

3.4.1.13 - CRawHeader Class

Class Hierarchy

```
CRawHeader
```

C++

```
class CRawHeader;
```

Description

This class represents the most basic form of SIP header. This is composed of the header name and the header value. This class handles the unparsed form of a header, directly received from the network.

Reception of RAW data

The CRawHeader is able to cope with the reception of a stream of bytes that will eventually be split into single headers (CRLF-terminated lines). The AppendRawData (see page 220)() method can be used repeatedly to handle the buffering of bytes up to the CRLF. The IsComplete (see page 221)() method notifies the user if the Raw header considers it has enough data to be useable.

Location

SipParser/CRawHeader.h

Constructors

Constructor	Description
 CRawHeader (see page 219)	Constructor.

Legend

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

Destructors

Destructor	Description
 ~CRawHeader (see page 220)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 222)	Assignment operator.

Legend

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

Methods

Method	Description
 AppendRawData (see page 220)	Buffers characters of the header's name and body, up to the terminating CRLF.
 GetBody (see page 221)	Provides access to the header's body.
 GetName (see page 221)	Provides access to the header's name.
 IsComplete (see page 221)	Returns true if the raw header is considered complete, and as such futher calls to AppendRawData (see page 220) will fail.
 Reset (see page 222)	Returns the object to initial state.

Legend

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

3.4.1.13.1 - Constructors

3.4.1.13.1.1 - CRawHeader

3.4.1.13.1.1.1 - CRawHeader::CRawHeader Constructor

Constructor.

C++

```
CRawHeader();
```

Description

Default constructor.

3.4.1.13.1.1.2 - CRawHeader::CRawHeader Constructor

Copy Constructor.

C++

```
CRawHeader(IN const CRawHeader& rSrc);
```

Parameters

Parameters	Description
IN const CRawHeader& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.13.2 - Destructors**3.4.1.13.2.1 - CRawHeader::~CRawHeader Destructor**

Destructor.

C++

```
virtual ~CRawHeader();
```

Description

Destructor.

3.4.1.13.3 - Methods**3.4.1.13.3.1 - CRawHeader::AppendRawData Method**

Buffers characters of the header's name and body, up to the terminating CRLF.

C++

```
mxt_result AppendRawData(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the start of the data to append. It is adjusted as parsing occurs. See return codes for the value of rpcPos at method's return.

Returns

resSW_SIPPARSER_RAWHEADER_INCOMPLETE : All data copied, no CRLF found. rpcPos points to end of data '0'.

resSI_SIPPARSER_RAWHEADER_END_FOUND : Terminating CRLF found, data committed. rpcPos points to after the CRLF.

resFE_INVALID_STATE : Raw header already contains data. rpcPos untouched.

resFE_UNEXPECTED : Raw header only has a name, ':' is missing.

Description

Buffers data of a raw header received in streamed form. The buffered data must include the header name and body, and eventually terminating CRLF. Note that the Raw Headers can only be used in the context of a SIP header, meaning that the name must be followed by a colon and the body can be empty. This method handles the extension-header ABNF construct.

This method can be called as many times as necessary, up to the point where it finds the terminating CRLF.

When a line terminating CRLF is found in the data, the Commit() method is automatically called. Note that a CRLF followed by a terminating NULL does not assert that the CRLF is not part of LWS (the next AppendRawData will show if it was LWS or the end of the data).

This method is used during the CSipPacketParser (see page 312)'s AppendRawData routine only. Refer to its documentation for more details.

As stated earlier, CRawheader::AppendRawData is only used by SipPacketParser. Commit() chunks the raw data into raw headers. CSipUri (see page 325) does not use the raw data chunking capability because you can't have a CRLF in the middle of a SIPURI's headers. The need for the complex CRawheader buffering arises from the LWS structure's allowance of CRLF anywhere.

```
RFC 3261 ABNF:
extension-header = header-name HCOLON header-value
header-name     = token
header-value    = *(TEXT-UTF8char / UTF8-CONT / LWS)
message-body    = *OCTET
```

See Also

Commit, IsComplete (see page 221)

3.4.1.13.3.2 - GetBody**3.4.1.13.3.2.1 - CRawHeader::GetBody Method**

Provides access to the header's body.

C++

```
const CString& GetBody() const;
CString& GetBody();
```

Returns

Body of the header.

Description

Provides access to the header's body.

See Also

GetName (see page 221)

3.4.1.13.3.3 - GetName**3.4.1.13.3.3.1 - CRawHeader::GetName Method**

Provides access to the header's name.

C++

```
const CToken& GetName() const;
CToken& GetName();
```

Returns

Name of the header.

Description

Provides access to the header's name.

See Also

GetBody (see page 221)

3.4.1.13.3.4 - CRawHeader::IsComplete Method

Returns true if the raw header is considered complete, and as such further calls to AppendRawData (see page 220) will fail.

C++

```
bool IsComplete() const;
```

Returns

True if the header is considered complete. This means that the header has a non-empty name.

Description

Checks if the header is complete.

See Also

AppendRawData (see page 220)

3.4.1.13.3.5 - CRawHeader::Reset Method

Returns the object to initial state.

C++

```
void Reset();
```

Description

Resets the name and body data members.

3.4.1.13.4 - Operators**3.4.1.13.4.1 - CRawHeader::= Operator**

Assignment operator.

C++

```
CRawHeader& operator =(IN const CRawHeader& rSrc);
```

Parameters

Parameters	Description
IN const CRawHeader& rSrc	Source from which to copy.

Returns

Reference to "this" instance.

Description

Default assignment operator.

3.4.1.14 - CReginfo Class**Class Hierarchy****C++**

```
class CReginfo : private IXmlParserMgr;
```

Description

This class parses, stores, and generates a reginfo document as per RFC 3680 and extended by draft-ietf-sipping-gruu-reg-event-08.

When parsing, this class ignores namespaces. When generating, this class defines "urn:ietf:params:xml:ns:reginfo" as the default namespace.

If there is at least one "pub-gruu" or "temp-gruu" element defined, the generator defines "urn:ietf:params:xml:ns:gruuinfo" as a namespace with value 'gr'.

Location

SipParser/CReginfo.h

See Also

CRegistrationElement, CContactElement

Constructors

Constructor	Description
 CReginfo (see page 229)	Default Constructor.

Legend

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

Destructors

Destructor	Description
 ~CReginfo (see page 229)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 AppendRegistration (see page 230)	Appends a registration element.
 GetRegistration (see page 230)	Gets the specified registration element.
 GetRegistrationCount (see page 230)	Gets the number of registration.
 GetState (see page 231)	Returns the state attribute for this reginfo document.
 GetVersion (see page 231)	Gets the version attribute.
 Parse (see page 231)	Parses a reginfo XML document.
 Reset (see page 232)	Resets internal settings.
 Serialize (see page 232)	Outputs the reginfo to the blob.
 SetState (see page 233)	Sets the state attribute for this reginfo document.
 SetVersion (see page 233)	Sets the version attribute.

Legend

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

Structs

Struct	Description
SContact (see page 223)	
SRegistration (see page 227)	

3.4.1.14.1 - Structs

3.4.1.14.1.1 - CReginfo::SContact Struct

Class Hierarchy

C++

```
struct SContact {
    CString m_strState;
    CString m_strEvent;
    unsigned int m_uDurationRegistereds;
    unsigned int m_uExpiresS;
    unsigned int m_uRetryAfterS;
    CString m_strId;
    CString m_strQ;
    CString m_strCallId;
    unsigned int m_ucSeq;
    CNameAddr m_nameAddr;
    CString m_strInstanceId;
    IUri* m_pTempGruu;
    IUri* m_pPubGruu;
    CGenParamList m_unknownParams;
};
```

Description

Contains the information about a contact.

Constructors

Constructor	Description
 SContact (see page 227)	Constructor.

Legend

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

Destructors

Destructor	Description
 ~SContact (see page 227)	Destructor.

Legend

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

Methods

Method	Description
 IsValid (see page 227)	Validates the content of the Contact.

Legend

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

3.4.1.14.1.1.1 - Data Members

3.4.1.14.1.1.1.1 - CReginfo::SContact::m_nameAddr Data Member

The name addr associated with that contact.

C++

```
CNameAddr m_nameAddr;
```

Description

The URI part of the name addr is the "uri" element and the display-name part is the "display-name" element. If the display-name is empty, the "display-name" element is omitted.

3.4.1.14.1.1.1.2 - CReginfo::SContact::m_pPubGruu Data Member

The public GRUU associated with the combination of the AOR and the Instance ID.

C++

```
IUri* m_pPubGruu;
```

Description

This parameter is optional and SHOULD be included only when the instance-id is included. It corresponds to a "pub-gruu" element of the contact. The structure has the ownership of this object. When the pointer is NULL, the element is not part of the contact element.

3.4.1.14.1.1.1.3 - CReginfo::SContact::m_pTempGruu Data Member

Temporary GRUU associated with the combination of the AOR and the Instance ID.

C++

```
IUri* m_pTempGruu;
```

Description

This parameter is optional and MAY be included only when the instance-id is included. It corresponds to a "temp-gruu" element of the contact. The structure has the ownership of this object. When the pointer is NULL, the element is not part of the contact element.

3.4.1.14.1.1.1.4 - CReginfo::SContact::m_strCallId Data Member

The current Call-ID carried in the REGISTER that was last used to update this contact.

C++

```
CString m_strCallId;
```

Description

This parameter is optional and is not used when empty.

3.4.1.14.1.1.1.5 - CReginfo::SContact::m_strEvent Data Member

The event that caused the contact state machine to go into its current state.

C++

```
CString m_strEvent;
```

Description

The valid values are "created", "deactivated", "expired", "probation", "refreshed", "registered", "rejected", "shortened", or "unregistered", but this is not enforced by this object. This parameter is mandatory.

3.4.1.14.1.1.1.6 - CReginfo::SContact::m_strId Data Member

The id for that contact.

C++

```
CString m_strId;
```

Description

It should be the same across notification if the URI is the same. This parameter is mandatory.

3.4.1.14.1.1.1.7 - CReginfo::SContact::m_strInstanceId Data Member

The Instance ID associated with this contact.

C++

```
CString m_strInstanceId;
```

Description

This parameter is optional and is not used when empty. It corresponds to an "unknown-param" element with a 'name' attribute of "+sip.instance".

3.4.1.14.1.1.1.8 - CReginfo::SContact::m_strQ Data Member

The q parameter of the contact.

C++

```
CString m_strQ;
```

Description

This parameter is optional and is not used when empty. This parameter should be a float but this rule is not enforced by this object.

3.4.1.14.1.1.1.9 - CReginfo::SContact::m_strState Data Member

Values can be "active" or "terminated" but this is not enforced by this object.

C++

```
CString m_strState;
```

Description

This parameter is mandatory.

3.4.1.14.1.1.1.10 - CReginfo::SContact::m_uCSeq Data Member

The last CSeq value present in a REGISTER request that updated this contact value.

C++

```
unsigned int m_uCSeq;
```

Description

This parameter is optional. To omit it, set it to `uREGINFO_CONTACT_INVALID_CSEQ` (see page 401), which is also the default value.

3.4.1.14.1.1.1.11 - CReginfo::SContact::m_uDurationRegisteredS Data Member

The amount of time that the contact has been bound to the address-of-record, in seconds.

C++

```
unsigned int m_uDurationRegisteredS;
```

Description

This parameter is optional. To omit it, set it to `uREGINFO_CONTACT_INVALID_DURATION_REGISTERED` (see page 401), which is also the default value.

3.4.1.14.1.1.1.12 - CReginfo::SContact::m_uExpiresS Data Member

The number of seconds remaining until the binding is due to expire.

C++

```
unsigned int m_uExpiresS;
```

Description

This parameter is mandatory when the event is "shortened", otherwise, it is optional when the state is "active" and finally, it is unused otherwise. Note however that this object does not enforce these rules. To omit it, set it to `uREGINFO_CONTACT_INVALID_EXPIRES` (see page 401), which is also the default value.

3.4.1.14.1.1.1.13 - CReginfo::SContact::m_unknownParams Data Member

The unknown parameters associated with the contact.

C++

```
CGenParamList m_unknownParams;
```

Description

There can be 0 or more unknown parameters. These parameters each correspond to one "unknown-param" element with the parameter name as the "name" attribute and the parameter value as the element value. Note that the CGenParamList (see page 146) is used solely as a container and that it cannot be used to parse and generate the XML elements. Also note that the elements already explicitly mentioned in this structure (like 'q' or "+sip.instance") should not be duplicated in this list.

3.4.1.14.1.1.14 - CReginfo::SContact::m_uRetryAfterS Data Member

The amount of seconds after which the owner of the contact is expected to retry its registration.

C++

```
unsigned int m_uRetryAfterS;
```

Description

This parameter is mandatory when the event is "probation". Otherwise, it is not used. Note however that this object does not enforce these rules. To omit it, set it to uREGINFO_CONTACT_INVALID_RETRY_AFTER (see page 401), which is also the default value.

3.4.1.14.1.1.2 - Constructors

3.4.1.14.1.1.2.1 - CReginfo::SContact::SContact Constructor

Constructor.

C++

```
SContact();
```

3.4.1.14.1.1.3 - Destructors

3.4.1.14.1.1.3.1 - CReginfo::SContact::~SContact Destructor

Destructor.

C++

```
~SContact();
```

3.4.1.14.1.1.4 - Methods

3.4.1.14.1.1.4.1 - CReginfo::SContact::IsValid Method

Validates the content of the Contact.

C++

```
bool IsValid() const;
```

3.4.1.14.1.2 - CReginfo::SRegistration Struct

Class Hierarchy

C++

```
struct SRegistration {
    IUri* m_pAor;
    CString m_strId;
    CString m_strState;
    CVecto<SContact*> m_vecpstContacts;
};
```

Description

Contains the information about a registration.

Constructors

Constructor	Description
♦ SRegistration (see page 229)	Constructor.

Legend

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

Destructors

Destructor	Description
 ~SRegistration (see page 229)	Destructor.

Legend

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

Methods

Method	Description
 IsValid (see page 229)	Validates the content of the Registration.

Legend

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

3.4.1.14.1.2.1 - Data Members**3.4.1.14.1.2.1.1 - CReginfo::SRegistration::m_pAor Data Member**

The URI, which is the address-of-record this registration refers to.

C++

```
IUri* m_pAor;
```

Description

This is the "aor" attribute of the "registration" element. This parameter is mandatory. The structure has the ownership of this object.

3.4.1.14.1.2.1.2 - CReginfo::SRegistration::m_strId Data Member

The id attribute identifies this registration.

C++

```
CString m_strId;
```

Description

This parameter is mandatory.

3.4.1.14.1.2.1.3 - CReginfo::SRegistration::m_strState Data Member

Indicates the state of the registration.

C++

```
CString m_strState;
```

Description

The valid values are "init", "active", and "terminated" but this is not enforced by this object. This parameter is mandatory.

3.4.1.14.1.2.1.4 - CReginfo::SRegistration::m_vecpstContacts Data Member

The registered contacts.

C++

```
CVector<SContact*> m_vecpstContacts;
```

Description

There can be any number of contacts (including 0) and they each correspond to one "contact" element.

3.4.1.14.1.2.2 - Constructors**3.4.1.14.1.2.2.1 - CReginfo::SRegistration::SRegistration Constructor**

Constructor.

C++

```
SRegistration();
```

3.4.1.14.1.2.3 - Destructors**3.4.1.14.1.2.3.1 - CReginfo::SRegistration::~SRegistration Destructor**

Destructor.

C++

```
~SRegistration();
```

3.4.1.14.1.2.4 - Methods**3.4.1.14.1.2.4.1 - CReginfo::SRegistration::IsValid Method**

Validates the content of the Registration.

C++

```
bool IsValid() const;
```

3.4.1.14.2 - Constructors**3.4.1.14.2.1 - CReginfo::CReginfo Constructor**

Default Constructor.

C++

```
CReginfo();
```

Description

Constructor.

3.4.1.14.3 - Destructors**3.4.1.14.3.1 - CReginfo::~CReginfo Destructor**

Destructor.

C++

```
virtual ~CReginfo();
```

Description

Destructor.

3.4.1.14.4 - Methods

3.4.1.14.4.1 - CReginfo::AppendRegistration Method

Appends a registration element.

C++

```
void AppendRegistration(IN TO SRegistration* pstRegistration);
```

Parameters

Parameters	Description
IN TO SRegistration* pstRegistration	A pointer to the specified registration. Ownership is taken.

Description

Appends the registration element in parameter to this reginfo document. Ownership of the parameter is taken. Calling this method with a NULL pointer has no effect.

3.4.1.14.4.2 - GetRegistration

3.4.1.14.4.2.1 - CReginfo::GetRegistration Method

Gets the specified registration element.

C++

```
SRegistration* GetRegistration(IN unsigned int uIdx);
```

Parameters

Parameters	Description
IN unsigned int uIdx	The index of the registration to obtain.

Returns

A pointer to the specified registration. It is NULL if the index is invalid.

Description

Gets the specified registration element. Note that this pointer is short lived and should not be stored for future use.

3.4.1.14.4.2.2 - CReginfo::GetRegistration Method

Gets the specified registration element.

C++

```
const SRegistration* GetRegistration(IN unsigned int uIdx) const;
```

Parameters

Parameters	Description
IN unsigned int uIdx	The index of the registration to obtain.

Returns

A pointer to the specified registration. It is NULL if the index is invalid.

Description

Gets the specified registration element. Note that this pointer is short lived and should not be stored for future use.

3.4.1.14.4.3 - CReginfo::GetRegistrationCount Method

Gets the number of registration.

C++

```
unsigned int GetRegistrationCount() const;
```

Returns

The number of registrations.

Description

Gets the number of registrations in this reginfo XML document.

3.4.1.14.4.4 - CReginfo::GetState Method

Returns the state attribute for this reginfo document.

C++

```
const char* GetState() const;
```

Returns

The state attribute. It can be NULL when not set.

Description

Returns the state of this reginfo document. The state indicates whether the document contains the full registration state or only information on those registrations that have changed since the previous document (partial).

This attribute is mandatory. It is always output in generated reginfo documents.

3.4.1.14.4.5 - CReginfo::GetVersion Method

Gets the version attribute.

C++

```
unsigned int GetVersion() const;
```

Returns

The version attribute.

Description

Gets the version of the reginfo document. This version number starts at 0 and is incremented by 1 each time a new reginfo document is created inside a subscription. It enables the subscriber to properly order the received reginfo document and detect when there is a gap between two partial documents so the subscriber can request a new full document.

This attribute is mandatory and is always output in generated reginfo documents.

This attribute defaults to 0.

3.4.1.14.4.6 - Parse**3.4.1.14.4.6.1 - CReginfo::Parse Method**

Parses a reginfo XML document.

C++

```
mxt_result Parse(IN const CBlob& rDocument);
```

Parameters

Parameters	Description
IN const CBlob& rDocument	The document held within a blob object.

Returns

- resFE_INVALID_ARGUMENT: The data is not a valid XML document.
- resFE_FAIL: The data is not a valid reginfo document.
- resS_OK: Data successfully parsed.

Description

Parses the reginfo XML document. The parsing follows RFC 3680 and draft-ietf-sipping-gruu-reg-event-08. The comments, the unknown elements, and the unknown attributes are ignored. Note that the "unknown-param" element is NOT an unknow element.

3.4.1.14.4.6.2 - CReginfo::Parse Method

Parses a reginfo XML document.

C++

```
mxt_result Parse(IN unsigned int uDataSize, INOUT const uint8_t* puBuffer);
```

Parameters

Parameters	Description
IN unsigned int uDataSize	The size of the data to parse.
INOUT const uint8_t* puBuffer	The data to parse.

Returns

- resFE_INVALID_ARGUMENT: puBuffer is NULL.
- resFE_FAIL: The data is not a valid reginfo document or XML parsing failed.
- resS_OK: Data successfully parsed.

Description

Parses the reginfo XML document. The parsing follows RFC 3680 and draft-ietf-sipping-gruu-reg-event-08. The comments, the unknown elements, and the unknown attributes are ignored. Note that the "unknown-param" element is NOT an unknow element.

3.4.1.14.4.7 - CReginfo::Reset Method

Resets internal settings.

C++

```
void Reset();
```

Description

Resets internal settings.

3.4.1.14.4.8 - CReginfo::Serialize Method

Outputs the reginfo to the blob.

C++

```
mxt_result Serialize(INOUT CBlob& rBlob);
```

Parameters

Parameters	Description
INOUT Cblob& rBlob	The blob where to put the reginfo document.

Returns

- resFE_INVALID_STATE: The object is missing mandatory configuration.
- resFE_FAIL: Problems writing to the blob.
- resS_OK: The reginfo document is properly output to the blob.

Description

Outputs this reginfo document to the blob in parameter. The content of the blob is overidden with the reginfo document.

3.4.1.14.4.9 - CReginfo::SetState Method

Sets the state attribute for this reginfo document.

C++

```
void SetState(IN const char* pszState);
```

Parameters

Parameters	Description
szState	The state attribute.

Description

Sets the state of this reginfo document. The state indicates whether the document contains the full registration state or only information on those registrations that have changed since the previous document (partial).

The allowed values as per RFC 3680 are "partial" and "full" but this method does not enforce it.

This attribute is mandatory. It is always output in generated reginfo documents.

3.4.1.14.4.10 - CReginfo::SetVersion Method

Sets the version attribute.

C++

```
void SetVersion(IN unsigned int uVersion);
```

Parameters

Parameters	Description
IN unsigned int uVersion	The version attribute.

Description

Sets the version of the reginfo document. This version number starts at 0 and is incremented by 1 each time a new reginfo document is created inside a subscription. It enables the subscriber to properly order the received reginfo document and detect when there is a gap between two partial documents so the subscriber can request a new full document.

This attribute is mandatory and is always output in generated reginfo documents.

This attribute defaults to 0.

3.4.1.15 - CRequestLine Class

Class Hierarchy



C++

```
class CRequestLine;
```

Description

The CRequestLine class handles the storage, parsing, and serialization of the request-line construct. It is composed of a method and a Request-URI.

The M5T SIP stack does not store the SIP-Version. It only supports SIP/2.0.

```
RFC 3261 ABNF:
Request-Line = Method SP Request-URI SP SIP-Version CRLF
Request-URI = SIP-URI / SIPS-URI / absoluteURI
SIP-Version = "SIP" "/" 1*DIGIT "." 1*DIGIT
```

Location

SipParser/CRequestLine.h

See Also

Method.h

Constructors

Constructor	Description
 CRequestLine (see page 235)	Constructor.

Legend

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

Destructors

Destructor	Description
 ~CRequestLine (see page 235)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 238)	Assignment operator.

Legend

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

Methods

Method	Description
 GetMethod (see page 235)	Converts the method to its enum form.
 GetMethodToken (see page 236)	Provides access to the method data member.
 GetSipUri (see page 236)	Provides access to the Request-URI and casts it to CSipUri (see page 325) if applicable.
 GetUri (see page 236)	Provides access to the Request-URI.
 Parse (see page 237)	Parses the Request-Line.
 Reset (see page 237)	Resets the method and Request-URI.
 Serialize (see page 237)	Outputs the data members into the buffer.

• SetRequestUri (see page 238)	Sets the Request-URI. Takes ownership of the pointer.
• SetSipUri (see page 238)	Sets the Request-URI as a SIPURI.

Legend

•	Method
---	--------

3.4.1.15.1 - Constructors**3.4.1.15.1.1 - CRequestLine****3.4.1.15.1.1.1 - CRequestLine::CRequestLine Constructor**

Constructor.

C++

```
CRequestLine();
```

Description

Constructor.

3.4.1.15.1.1.2 - CRequestLine::CRequestLine Constructor

Copy constructor.

C++

```
CRequestLine(IN const CRequestLine& rSrc);
```

Parameters

Parameters	Description
IN const CRequestLine& rSrc	Source from which to copy.

Description

Copy constructor. Copies the method and Request-URI.

3.4.1.15.2 - Destructors**3.4.1.15.2.1 - CRequestLine::~CRequestLine Destructor**

Destructor.

C++

```
virtual ~CRequestLine();
```

Description

Destructor.

3.4.1.15.3 - Methods**3.4.1.15.3.1 - CRequestLine::GetMethod Method**

Converts the method to its enum form.

C++

```
ESipMethod GetMethod() const;
```

Returns

Method enum.

Description

Converts the method string to its enumerate equivalent.

See Also

[MxConvertSipMethod](#) (see page 390)

3.4.1.15.3.2 - **GetMethodToken**

3.4.1.15.3.2.1 - **CRequestLine::GetMethodToken Method**

Provides access to the method data member.

C++

```
const CToken& GetMethodToken() const;
CToken& GetMethodToken();
```

Returns

Constant reference on the method.

Description

Provides access to the method.

3.4.1.15.3.3 - **GetSipUri**

3.4.1.15.3.3.1 - **CRequestLine::GetSipUri Method**

Provides access to the Request-URI and casts it to CSipUri (see page 325) if applicable.

C++

```
inline const CSipUri* GetSipUri() const;
inline CSipUri* GetSipUri();
```

Returns

Constant pointer to the Request-URI, if it is a SIP or SIPS URI. It is NULL otherwise.

Description

Provides access to the Request-URI in the CSipUri (see page 325) form.

See Also

[SetSipUri](#) (see page 238), [GetUri](#) (see page 236), [SetUri](#)

3.4.1.15.3.4 - **GetUri**

3.4.1.15.3.4.1 - **CRequestLine::GetUri Method**

Provides access to the Request-URI.

C++

```
const IUri* GetUri() const;
IUri* GetUri();
```

Returns

Pointer to the Request-URI. It may be NULL.

Description

Provides access to the Request-URI.

See Also

SetUri, GetSipUri (see page 236), SetSipUri (see page 238)

3.4.1.15.3.5 - CRequestLine::Parse Method

Parses the Request-Line.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. If the method fails, the pointer is set to the position of the error. If the method succeeds, rpcPos points after the CRLF terminating the request-line.

Returns

resSI_SIPPARSER_DATACONT : Request-line parsed, more data follows.

resS_OK : Request-line parsed, end of data follows.

resFE_UNEXPECTED : The request-line URI is enclosed within angle brackets.

resFE_UNEXPECTED : The mandatory terminating CRLF could not be found.

resFE_SIPPARSER_STARTLINE_UNKNOWN_PROTOCOL : SIP-Version is not SIP/2.0.

resFE_UNEXPECTED : Could not find the method.

This method can also return any error mxt_result that CUriFactory::ParseUri (see page 379)() can return.

Description

Parses the request line. Trailing LWS are not skipped. The terminating CRLF is mandatory. This method validates that the SIP-Version is SIP/2.0. If the version is different, it is not supported by this SIP stack.

```
RFC 3261 ABNF:
Request-Line   = Method SP Request-URI SP SIP-Version CRLF
Request-URI    = SIP-URI / SIPS-URI / absoluteURI
SIP-Version   = "SIP" "/" 1*DIGIT "." 1*DIGIT
```

3.4.1.15.3.6 - CRequestLine::Reset Method

Resets the method and Request-URI.

C++

```
void Reset();
```

Description

Resets the method and Request-URI.

3.4.1.15.3.7 - CRequestLine::Serialize Method

Outputs the data members into the buffer.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT Cblob& rBlob	Output buffer.

Description

Outputs the internal data into the buffer. First outputs the method, then the request-URI, and after that outputs the hardcoded SIP-Version "SIP/2.0" and the terminating CRLF.

3.4.1.15.3.8 - CRequestLine::SetRequestUri Method

Sets the Request-URI. Takes ownership of the pointer.

C++

```
void SetRequestUri(IN TO IUri* pUri);
```

Parameters

Parameters	Description
IN TO IUri* pUri	URI to set. Ownership is taken.

Description

Sets the Request-URI. If a Request-URI is already set, it is released and replaced by the new Request-URI.

See Also

[GetUri](#) (see page 236), [SetSipUri](#) (see page 238), [GetSipUri](#) (see page 236)

3.4.1.15.3.9 - CRequestLine::SetSipUri Method

Sets the Request-URI as a SIPURI.

C++

```
void SetSipUri(IN const CString& rstrHost, IN uint16_t uPort = CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT,
IN const CString& rstrUser = CString(), IN CSipUri::ESecurityFlag eSecured = CSipUri::eUNSECURE);
```

Parameters

Parameters	Description
IN const CString& rstrHost	Hostname to set.
IN uint16_t uPort = CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT	Port to set.
IN const CString& rstrUser = CString()	Optional user name.
IN CSipUri::ESecurityFlag eSecured = CSipUri::eUNSECURE	Security setting, eSECURE means SIPS.

Description

Sets the Request-URI as a SipUri. If a URI is already configured, it is released and replaced by the new URI.

See Also

[GetSipUri](#) (see page 236), [SetRequestUri](#) (see page 238), [GetRequestUri](#)

3.4.1.15.4 - Operators**3.4.1.15.4.1 - CRequestLine::= Operator**

Assignment operator.

C++

```
CRequestLine& operator =(IN const CRequestLine& rSrc);
```

Parameters

Parameters	Description
IN const CRequestLine& rSrc	Source from which to copy.

Returns

A reference on "this" instance.

Description

Assignment operator. Copies the method and Request-URI.

3.4.1.16 - CSipHeader Class

Class: CSipHeader

Class Hierarchy



C++

```
class CSipHeader;
```

Description

The CSipHeader class is primarily responsible for parsing, serializing, and containing the header body. Each CSipHeader instance has a type (ESipHeaderType (see page 384)) that gives it enough information (by accessing the header definition table) to handle parsing and serialization for any header type.

The CSipHeader class applies very generic parsing rules. In particular, there is no semantic validation of the headers whatsoever. The only validation that takes place is the syntactic validation, i.e., parsing errors are flagged.

The CSipHeader class is also a single-linked chain of CSipHeaders. This functionality, referred to as the header chain, is used for storing multiple instances of the same header type. For example, two Contacts would be available through one CSipHeader. To access the second contact, the GetNextHeader (see page 265)() method would be used on the first contact. This functionality works equally for multi header that can be comma-separated or not (for example, many WWW-Authenticate headers can be found in a single packet, but cannot be combined on a single comma-separated line).

The CSipHeader class also contains the controls for header generation such as comma separation configuration and short/long header name form configuration. See CSipCoreConfig::SetCommaSeparatedHeader and CSipCoreConfig::SetHeaderFormPreference for more details.

Raw versus Parsed header

The CSipHeader class can contain two versions of the same data. The raw data is the data received from the network. The parsed data is the raw data after it has been processed.

A CSipHeader is considered as "raw header" when its data has not been parsed.

If the header type can be comma-separated, a single raw header can contain multiple headers. For example, a raw "Allow" header's data could look like this: "allow1, allow2". This raw header would be counted as one header, because until the raw data is parsed, the CSipHeader cannot know that the raw data actually contains two headers.

The header chain can be made up of parsed and raw headers, in any order. The basic use case consists in calling the Parse (see page 290)() method, after which the first headers of the chain are parsed, and the end of the chain (if not entirely parsed) will contain raw headers.

Refer to CSipPacketParser (see page 312) for more details on the parsing strategy.

Generic parsing structure

As stated before, the CSipHeader class generically handles any SIP header. At its core, a SIP header is made up of a name, a separator (such as ':'), a header-specific body, and an optional parameter list.

Since headers are contextual (can be found in the context of a SIP packet or SIP URI), the header name, name-body separator, and the terminating separator (CRLF for SIP packet, '=' for SIP URIs) are not handled in this class. As such, the CSipHeader is context-less.

CSipHeader handles the body of the header, which includes the header-specific data and the optional parameter list. It also handles comma-separated instances, since these do not have a header name in-between.

To take up the least space possible in RAM, the parsed data is stored in a generic byte array. Users of the CSipHeader don't actually need to know the details of this implementation, but do need to be warned that improper use of the CSipHeader interface (particularly any of the helpers for header body access such as GetAllow (see page 249), SetContactWildcard (see page 292), etc.) can lead to very strange problems. For example, using the SetAllow method when the header type is not eHDR_ALLOW can and will probably cause invalid data accesses.

With that said, the byte array is allocated when the CSipHeader is created. Using a byte array minimises memory fragmentation. When a new header is to be created, the easiest way to do so is by using the access helpers. These automatically create the correct data structures. In this case, the CSipHeader only contains parsed data, and no raw data.

Serialization and RAM use

When the CSipHeader contains raw data, it takes up this space in RAM. If that same header is parsed, the raw data still takes up the same RAM as before, and now the parsed version of the data also uses new RAM. For this reason, it is best to avoid parsing headers that are not really required by the application.

At serialization time, the CSipHeader first checks if it contains raw data. Regardless of the parsed data, if raw data is available, it is output into the buffer instead of the parsed data. The parsed data is serialized only if no raw data is available.

The reasons for this behaviour are twofold. When raw data is available, it is assumed that this raw data comes from the network. The first reason is that serializing the raw data is much less costly in execution, because it consists in a single memcpy. Serializing the parsed data is much longer because it breaks down into many function calls and memcpys. The second reason is that parsing the data may have altered it. For example, when parsing "allowrn ,allow2", the extra "rn " is lost. Serializing the raw data thus sends the header as it was received, which should help with interoperation scenarios.

The problem with serializing the raw data is that any change made by the user to the parsed data is ignored at Serialization time. It is thus UP TO THE USER to nullify the raw data if he wishes to use an existing header and modify it.

Warning

The user of the CSipHeader is required to verify the header type before using any of the Set/Get methods. In the interest of keeping the code size small, no verifications are made versus the header type or the current real use of the byte array data structure in these methods. It is thus VERY IMPORTANT that CSipHeader users make sure that they are using the right kind of interface!

Header accessor nomenclature

Access to each header's internal data is supplied through simple methods. Each header has a method to get and set each of its internal data constructs as per the appropriate ABNF. Note that there are no specific Get/Set methods for the parameter lists (usually described as *(SEMI param) in the ABNF) as this structure is generically handled by the CSipHeader's m_pParamList data member (through the CSipHeader::GetParamList (see page 267)() methods).

The header access methods are grouped by RFC/draft document.

Get and Set methods are formed as such: Get(HeaderName)(DataItem) where:

- (HeaderName) is the header name, such as Allow or Reply-To;
- (DataItem) is the name of the item to access. If there is only one item in the header's ABNF, then the (DataItem) part is omitted in the method name.

Location

SipParser/CSipHeader.h

Example 1

```
RFC 3261 ABNF:
Allow = "Allow" HCOLON [Method *(COMMA Method)]
```

The Allow header only has one data item (Method). As such, its accessors are:

- GetAllow (see page 249), SetAllow

Example 2

```
RFC 3262 ABNF:
RAck      = "RAck" HCOLON response-num LWS CSeq-num LWS Method
response-num = 1*DIGIT
CSeq-num   = 1*DIGIT
```

The RAck header has three data items. As such, its accessors are:

- GetRackCSeq (see page 273), GetRackMethod (see page 274), GetRackResponseNum (see page 274)

Example 3

```
RFC 3261 ABNF:
Reply-To      = "Reply-To" HCOLON rplyto-spec
rplyto-spec   = ( name-addr / addr-spec )
                  *( SEMI rplyto-param )
rplyto-param  = generic-param
```

The Reply-To header has one data item, plus a parameter list. As such, its accessors are:

- GetReplyTo (see page 278) (const and non-const)

These both access the (name-addr / addr-spec) construct (not the rplyto-spec construct), since the *(SEMI rplyto-param) construct is handled by the generic parameter list m_pParamList.

Example 4

```
RFC 3261 ABNF:
Proxy-Authenticate = "Proxy-Authenticate" HCOLON challenge
challenge         = ("Digest" LWS digest-cln *(COMMA digest-cln))
                  / other-challenge
other-challenge   = auth-scheme LWS auth-param
                  *(COMMA auth-param)
```

The authentication headers are special cases, as they all contain an auth-scheme, followed by a parameter list that does not start with a SEMI. In those cases, all parameters have been put into the parameter list (m_pParamList), as usual. To avoid any confusion, the accessors for these headers specify that the auth-scheme is fetched:

- GetAuthorizationScheme (see page 250)
- GetWwwAuthenticateScheme (see page 288)
- GetProxyAuthenticateScheme (see page 272)
- GetProxyAuthorizationScheme (see page 272)

Example 5

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

After this header is parsed, it contains a chain of headers. The first contains a CToken (see page 360) with "INVITE" for data. The second CSipHeader contains a CToken (see page 360) with "ACK" for data, and the third a CToken (see page 360) with "OPTIONS" for data, etc. Calling GetAllow (see page 249) right after the header is parsed returns a reference to the CToken (see page 360) with "INVITE" for data. To have access to the following CSipHeader in the chain, it is necessary to use GetNextHeader (see page 265) followed by GetAllow (see page 249) to access the CTokens.

Example 6

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

After this header is parsed, it contains a chain of headers. The first contains a CToken (see page 360) with "da" for data. The second CSipHeader contains a CToken (see page 360) with "en-gb" for data and a parameter list with a parameter "q" set to "0.8", and the third a CToken (see page 360) with "en" for data and a parameter "q" set to "0.7". Calling GetAcceptLanguage (see page 247) right after the header is parsed returns a reference to the CToken (see page 360) with "da" for data. To have access to the following CSipHeader in the chain, it is necessary to use GetNextHeader (see page 265) followed by GetAcceptLanguage (see page 247) to access the CTokens or GetParamList (see page 267) to access the parameter list.

Example 7

```
Call-Info: <http://www.example.com/alice/photo.jpg>;purpose=icon,
<http://www.example.com/alice/>;purpose=info
```

After this header is parsed, it contains a chain of headers. The first contains a CNameAddr (see page 201) with URI

"http://www.example.com/alice/photo.jpg" and a parameter list with the parameter purpose set to "icon". The second CSipHeader contains a CNameAddr (see page 201) with URI "http://www.example.com/alice/" and a parameter list with the parameter purpose set to "info". Calling GetCallId (see page 251) right after the header is parsed returns a reference to the CNameAddr (see page 201) with "http://www.example.com/alice/photo.jpg" for URI. To have access to the following CSipHeader in the chain, it is necessary to use GetNextHeader (see page 265) followed by GetCallId (see page 251) to access the second CNameAddress or GetParamList (see page 267) to access the parameter list.

Constructors

Constructor	Description
CSipHeader (see page 245)	Constructor. Takes any type of ESipHeaderType (see page 384).

Legend

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

Destructors

Destructor	Description
~CSipHeader (see page 245)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
!= (see page 296)	Comparison operator.
= (see page 296)	Assignment operator.
== (see page 296)	Comparison operator.

Legend

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

Methods

Method	Description
AppendNextHeader (see page 246)	Inserts a header at the end of the chain.
CopySingleHeader (see page 246)	Copies this header without its next headers.
GetAcceptContact (see page 246)	Provides access to the Accept-Contact header's "*" construct.
GetAcceptEncoding (see page 247)	Provides access to the Accept-Encoding header's content-coding construct.
GetAcceptLanguage (see page 247)	Provides access to the Accept-Language header's language-range construct.
GetAcceptMSubType (see page 248)	Provides access to the Accept header's m-subtype construct.
GetAcceptMType (see page 248)	Provides access to the Accept header's m-type construct.
GetAcceptResourcePriority (see page 248)	Provides access to the Accept-Resource-Priority header's [Resource-value] construct.
GetAlertInfo (see page 249)	Provides access to the Alert-Info header's absoluteURI construct.
GetAllow (see page 249)	Provides access to the Allow header's Method construct.
GetAllowEvents (see page 250)	Provides access to the Allow-Events header's event-type construct.
GetAuthenticationInfo (see page 250)	Provides access to the Authentication-Info header's ainfo construct.
GetAuthorizationScheme (see page 250)	Provides access to the Authorization header's auth-scheme construct.
GetCallId (see page 251)	Provides access to the Call-Id header's callid construct.
GetCallInfo (see page 251)	Provides access to the Call-Info header's absoluteURI construct.
GetContact (see page 252)	Provides access to the Contact header's (name-addr / addr-spec) construct.
GetContentDescription (see page 252)	Provides access to the Content-Description header's text construct.
GetContentDisposition (see page 252)	Provides access to the Content-Disposition header's disp-type construct.
GetContentEncoding (see page 253)	Provides access to the Content-Encoding header's content-coding construct.
GetContentId (see page 253)	Provides access to the Content-ID header's addr-spec (RFC 2822) construct.
GetContentLanguage (see page 254)	Provides access to the Content-Language header's language-tag construct.
GetContentLength (see page 254)	Provides access to the Content-Length header's 1*DIGIT construct.
GetContentTransferEncoding (see page 254)	Provides access to the Content-Language header's mechanism construct.

• GetContentTypeMSubType (See page 255)	Provides access to the Content-Type header's m-subtype construct.
• GetContentTypeMType (See page 255)	Provides access to the Content-Type header's m-type construct.
• GetCSeqMethod (See page 256)	Provides access to the CSeq header's Method construct.
• GetCSeqNumber (See page 256)	Provides access to the CSeq header's 1*DIGIT construct.
• GetDate (See page 256)	Provides access to the Date header's SIP-date construct.
• GetDiversion (See page 257)	Provides access to the Diversion header's name-addr.
• GetErrorInfo (See page 256)	Provides access to the Date header's SIP-date construct.
• GetEvent (See page 257)	Provides access to the Event header's event-type construct.
• GetExpires (See page 258)	Provides access to the Expires header's delta-seconds construct.
• GetExtensionHeaderName (See page 258)	Provides access to the Extension-header's header-name construct.
• GetExtensionHeaderValue (See page 259)	Provides access to the Extension-header's header-value construct.
• GetFlowTimer (See page 258)	Provides access to the Expires header's delta-seconds construct.
• GetFrom (See page 259)	Provides access to the From header's from-spec construct.
• GetHeaderName (See page 260)	Retrieves the long or short form following ShortNameForm configuration.
• GetHeaderType (See page 260)	Returns the header type.
• GetHistoryInfo (See page 260)	Provides access to the History-Info header's hi-targeted-to-uri construct.
• GetIdentity (See page 261)	Provides access to the Identity header's signed-identity-digest construct.
• GetIdentityInfo (See page 261)	Provides access to the Identity-Info header's ident-info construct.
• GetInReplyTo (See page 261)	Provides access to the In-Reply-To header's callid construct.
• GetJoin (See page 262)	Provides access to the Join header's callid construct.
• GetLongHeaderName (See page 262)	Retrieves the long form header name for this header.
• GetMaxForwards (See page 263)	Provides access to the Max-Forwards header's 1*DIGIT construct.
• GetMimeVersion (See page 263)	Provides access to the MIME-Version header's "1*DIGIT ." "1*DIGIT" construct.
• GetMinExpires (See page 263)	Provides access to the Min-Expires header's delta-seconds construct.
• GetMinSe (See page 264)	Provides access to the Min-SE header's delta-seconds construct.
• GetNbNextHeaders (See page 264)	Returns the number of headers in the chain, but omits the current header.
• GetNbParsedHeaders (See page 264)	Returns the number of parsed headers in the chain, including the current header.
• GetNextHeader (See page 265)	Returns a pointer to the chained header at the given 0-based index.
• GetOrganization (See page 265)	Provides access to the Organization header's [TEXT-UTF8-TRIM] construct.
• GetPAccessNetworkInfo (See page 266)	Provides access to the P-Access-Network-Info header's access-type construct.
• GetParam (See page 266)	Gets a header parameter value from the parameter list.
• GetParamList (See page 267)	Provides access to the parameter list.
• GetPAssertedIdentity (See page 268)	Provides access to the P-Asserted-Identity header's PAssertedID-value construct.
• GetPAssociatedUri (See page 268)	Provides access to the P-Associated-URI header's name-addr construct.
• GetPath (See page 268)	Provides access to the Path header's name-addr construct.
• GetPCalledPartyId (See page 269)	Provides access to the P-Called-Party-ID header's name-addr construct.
• GetPChargingFunctionAddresses (See page 269)	Provides access to the P-Charging-Function-Addresses header's mandatory first charge-addr-params construct. Other charge-addr-params are accessible through the parameter list.
• GetPChargingVector (See page 270)	Provides access to the P-Charging-Vector header's icid-value construct.
• GetPMediaAuthorization (See page 270)	Provides access to the P-Media-Authorization header's P-Media-Authorization-Token construct.
• GetPPREFERREDIdentity (See page 270)	Provides access to the P-Preferred-Identity header's PPreferredID-value construct.
• GetPriority (See page 271)	Provides access to the Priority header's priority-value construct.
• GetPrivacy (See page 271)	Provides access to the Privacy header's priv-value construct.
• GetProxyAuthenticateScheme (See page 272)	Provides access to the Proxy-Authenticate header's auth-scheme construct.
• GetProxyAuthorizationScheme (See page 272)	Provides access to the Proxy-Authorization header's auth-scheme construct.
• GetProxyRequire (See page 273)	Provides access to the Proxy-Require header's option-tag construct.
• GetPVisitedNetworkId (See page 273)	Provides access to the P-Visited-Network-ID header's (token / quoted-string) construct.
• GetRackCSeq (See page 273)	Provides access to the RAck header's CSeq-num construct.
• GetRackMethod (See page 274)	Provides access to the RAck header's Method construct.
• GetRackResponseNum (See page 274)	Provides access to the RAck header's response-num construct.
• GetRawHeader (See page 274)	Provides access to the raw header.
• GetReason (See page 275)	Provides access to the Reason header's protocol construct.
• GetRecordRoute (See page 275)	Provides access to the Record-Route header's name-addr construct.

• GetReferredBy (see page 276)	Provides access to the Referred-By header's referrer-uri construct.
• GetReferTo (see page 276)	Provides access to the Refer-To header's a(name-addr / addr-spec) construct.
• GetRejectContact (see page 277)	Provides access to the Reject-Contact header's "*" construct.
• GetReplaces (see page 277)	Provides access to the Replaces header's callid construct.
• GetReplyTo (see page 278)	Provides access to the Reply-To header's (name-addr / addr-spec) construct.
• GetRequestDisposition (see page 278)	Provides access to the Request-Disposition header's directive construct.
• GetRequire (see page 278)	Provides access to the Require header's option-tag construct.
• GetResourcePriority (see page 279)	Provides access to the Resource-Priority header's Resource-value construct.
• GetRetryAfter (see page 279)	Provides access to the Retry-After header's delta-seconds construct.
• GetRetryAfterComment (see page 280)	Provides access to the Retry-After header's comment construct.
• GetRoute (see page 280)	Provides access to the Route header's name-addr construct.
• GetRSeq (see page 280)	Provides access to the RSeq header's response-num construct.
• GetServer (see page 281)	Provides access to the Server header's server-val construct.
• GetServiceRoute (see page 281)	Provides access to the Service-Route header's name-addr construct.
• GetSessionExpires (see page 281)	Provides access to the Session-Expires header's delta-seconds construct.
• GetShortHeaderName (see page 282)	Retrieves the short form header name for this header, if available.
• GetSipETag (see page 282)	Provides access to the SIP-ETag header's entity-tag construct.
• GetSipIfMatch (see page 283)	Provides access to the SIP-If-Match header's entity-tag construct.
• GetSubject (see page 283)	Provides access to the Subject header's [TEXT-UTF8-TRIM] construct.
• GetSubscriptionState (see page 283)	Provides access to the Subscription-State header's substate-value construct.
• GetSupported (see page 284)	Provides access to the Supported header's [option-tag] construct.
• GetTargetDialog (see page 284)	Provides access to the Target-Dialog header's callid construct.
• GetTimestamp (see page 285)	Provides access to the Timestamp header's 1*(DIGIT) ["." *(DIGIT)] [LWS delay] construct.
• GetTo (see page 285)	Provides access to the To header's (name-addr / addr-spec) construct.
• GetUnsupported (see page 285)	Provides access to the Unsupported header's option-tag construct.
• GetUserAgent (see page 286)	Provides access to the User-Agent header's server-val construct.
• GetViaProtocolName (see page 286)	Provides access to the Via header's protocol-name construct.
• GetViaProtocolVersion (see page 287)	Provides access to the Via header's protocol-version construct.
• GetViaSentBy (see page 287)	Provides access to the Via header's sent-by construct.
• GetViaTransport (see page 287)	Provides access to the Via header's transport construct.
• GetWarning (see page 288)	Provides access to the Warning header's warning-value construct.
• GetWwwAuthenticateScheme (see page 288)	Provides access to the Www-Authenticate header's auth-scheme construct.
• InsertNextHeader (see page 288)	Inserts a header at the given 0-based index in the chain. Index 0 refers to this CSipHeader's first next header.
• IsContactWildcard (see page 289)	Returns true if the Contact header's value is STAR.
• IsEmptyHeader (see page 289)	Returns true if this header is empty, and can be empty as per the header definition table.
• IsParsedDataAvailable (see page 290)	Returns true if parsed data (parsed from raw or set by application) is available.
• IsSingleHdrEquivalent (see page 290)	Returns true if the current instance is equivalent to the cited instance, without regard to the next headers.
• Parse (see page 290)	Parses the header chain for up to uMaxHeaders.
• RemoveNextHeader (see page 291)	Removes a header from the chain, at the given 0-based index. Index 0 refers to this CSipHeader's first next header.
• ReplaceNextHeader (see page 292)	Replaces the next headers with the headers in parameter.
• Reset (see page 292)	Clears all parsed/Set data.
• Serialize (see page 292)	Outputs the body of the header into the blob.
• SetContactWildcard (see page 292)	Sets the value of the STAR for the Contact header's (name-addr / addr-spec) construct.
• SetNameForm (see page 293)	Sets the form of header name to use for all header types.
• SetParam (see page 293)	Sets a header parameter value to the parameter list from a CString.
• SetParamList (see page 294)	Sets the parameter list.
• SetRawHeader (see page 294)	Sets the raw header. Ownership is taken.
• UnlinkNextHeader (see page 295)	Removes a header from the chain and returns it, at the given 0-based index. Index 0 refers to this CSipHeader's first next header.
• UnlinkTopHeader (see page 295)	Unlinks the top header from the chain. The returned header is the chain excluding the current header.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
EHeaderNameForm (see page 297)	
EParameterCreationBehavior (see page 297)	
EResetLevel (see page 297)	

3.4.1.16.1 - Constructors**3.4.1.16.1.1 - CSipHeader****3.4.1.16.1.1.1 - CSipHeader::CSipHeader Constructor**

Constructor. Takes any type of ESipHeaderType (see page 384).

C++

```
CSipHeader(IN ESipHeaderType eHeader);
```

Parameters

Parameters	Description
IN ESipHeaderType eHeader	Type of header. It can be eHDR_EXTENSION.

Description

Constructor.

3.4.1.16.1.1.2 - CSipHeader::CSipHeader Constructor

Copy constructor.

C++

```
CSipHeader(IN const CSipHeader& rSrc);
```

Parameters

Parameters	Description
IN const CSipHeader& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.16.2 - Destructors**3.4.1.16.2.1 - CSipHeader::~CSipHeader Destructor**

Destructor.

C++

```
virtual ~CSipHeader();
```

Description

Destructor.

3.4.1.16.3 - Methods

3.4.1.16.3.1 - CSipHeader::AppendNextHeader Method

Inserts a header at the end of the chain.

C++

```
mxt_result AppendNextHeader(IN TO CSipHeader* pNewHeader, IN bool bTakeOwnershipOnSuccess = false);
```

Parameters

Parameters	Description
IN TO CSipHeader* pNewHeader	Header to add at the end of the list.
IN bool bTakeOwnershipOnSuccess = false	<ul style="list-style-type: none"> false: Default value. Ownership of pNewHeader is always taken. true: Ownership is taken if the return value is a success only.

Returns

resS_OK : Header is correctly appended.

resFE_INVALID_ARGUMENT : Given header is of a different type than current header.

resFE_INVALID_ARGUMENT : This type of header cannot have multiple instances within a same SIP packet.

Description

Inserts a header at the end of this header type's list. It can be used only for multi or combinable header types (e.g., Via, Contact, etc.). Only headers of the same type can be appended after each other. A chain of raw and parsed headers can be made.

See Also

RemoveNextHeader (see page 291)

3.4.1.16.3.2 - CSipHeader::CopySingleHeader Method

Copies this header without its next headers.

C++

```
void CopySingleHeader(IN const CSipHeader& rSrc);
```

Parameters

Parameters	Description
IN const CSipHeader& rSrc	The header to copy.

Description

Copies this header without its next headers.

The content of this header will be the same than the one of rSrc with the difference that it will have no next header regardless if rSrc has next headers or not.

See Also

operator=

3.4.1.16.3.3 - GetAcceptContact

3.4.1.16.3.3.1 - CSipHeader::GetAcceptContact Method

Provides access to the Accept-Contact header's "*" construct.

C++

```
const CToken& GetAcceptContact() const;
CToken& GetAcceptContact();
```

Returns

The token (CToken (see page 360)) in the Accept-Contact header field.

Description

Contains feature sets which, if matched by a UA, imply that the request should be routed to that UA. For example, in the header:

```
Accept-Contact: *;audio;require
```

this is the token containing "*".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.4 - GetAcceptEncoding**3.4.1.16.3.4.1 - CSipHeader::GetAcceptEncoding Method**

Provides access to the Accept-Encoding header's content-coding construct.

C++

```
const CToken& GetAcceptEncoding() const;
CToken& GetAcceptEncoding();
```

Returns

The token (CToken (see page 360)) in the Accept-Encoding header field.

Description

Restricts the content-codings that are acceptable in the response. For example, in the header:

```
Accept-Encoding: gzip
```

this is the token containing "gzip".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.5 - GetAcceptLanguage**3.4.1.16.3.5.1 - CSipHeader::GetAcceptLanguage Method**

Provides access to the Accept-Language header's language-range construct.

C++

```
const CToken& GetAcceptLanguage() const;
CToken& GetAcceptLanguage();
```

Returns

The token (CToken (see page 360)) in the Accept-Language header field.

Description

Indicates the preferred languages for reason phrases, session descriptions, or status responses carried as message bodies in the response. For example, in the header:

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

this is the token containing "da".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.6 - GetAcceptMSubType

3.4.1.16.3.6.1 - CSipHeader::GetAcceptMSubType Method

Provides access to the Accept header's m-subtype construct.

C++

```
const CToken& GetAcceptMSubType() const;
CToken& GetAcceptMSubType();
```

Returns

The second token (CToken (see page 360)) in the Accept header field..

Description

Indicates the sub-type of message body that the UA wants to receive in response. For example, in the header:

Accept: application/sdp

this is the token containing "sdp".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.7 - GetAcceptMType

3.4.1.16.3.7.1 - CSipHeader::GetAcceptMType Method

Provides access to the Accept header's m-type construct.

C++

```
const CToken& GetAcceptMType() const;
CToken& GetAcceptMType();
```

Returns

The first token (CToken (see page 360)) in the Accept header field.

Description

Indicates the type of message body that the UA wants to receive in response. For example, in the header:

Accept: application/sdp

this is the token containing "application".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.8 - GetAcceptResourcePriority

3.4.1.16.3.8.1 - CSipHeader::GetAcceptResourcePriority Method

Provides access to the Accept-Resource-Priority header's [Resource-value] construct.

C++

```
const CToken& GetAcceptResourcePriority() const;
CToken& GetAcceptResourcePriority();
```

Returns

The token (CToken (see page 360)) in the Accept-Resource-Priority header field.

Description

Enumerates the resource values (r-values) a SIP user agent server is willing to process. For example, in the header:

```
Accept-Resource-Priority: dsn.flash-override,  
dsn.flash, dsn.immediate, dsn.priority, dsn.routine
```

this is the token containing "dsn.flash-override".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.9 - GetAlertInfo

3.4.1.16.3.9.1 - CSipHeader::GetAlertInfo Method

Provides access to the Alert-Info header's absoluteURI construct.

C++

```
const CNameAddr& GetAlertInfo() const;  
CNameAddr& GetAlertInfo();
```

Returns

The token (CToken (see page 360)) in the Alert-Info header field.

Description

Specifies an alternative ring or ringback tone. For example, in the header:

```
Alert-Info: <http://www.example.com/sounds/moo.wav>  
  
this is the token containing  
<http://www.example.com/sounds/moo.wav> .
```

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.10 - GetAllow

3.4.1.16.3.10.1 - CSipHeader::GetAllow Method

Provides access to the Allow header's Method construct.

C++

```
const CToken& GetAllow() const;  
CToken& GetAllow();
```

Returns

The token (CToken (see page 360)) in the Allow header field.

Description

Lists the set of methods supported by the UA generating the message. For example, in the header:

```
Allow: INVITE, ACK, OPTIONS, CANCEL, BYE
```

this is the token containing "INVITE".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.11 - GetAllowEvents

3.4.1.16.3.11.1 - CSipHeader::GetAllowEvents Method

Provides access to the Allow-Events header's event-type construct.

C++

```
const CToken& GetAllowEvents() const;
CToken& GetAllowEvents();
```

Returns

The token (CToken (see page 360)) in the Allow-Events header field.

Description

Indicates the event packages supported by the UA. A UA sending an "Allow-Events" header is advertising that it can process SUBSCRIBE requests and generate NOTIFY requests for all of the event packages listed in that header. For example, in the header:

Allow-Events: spirits-INDPs

this is the token containing "spirits-INDPs".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.12 - GetAuthenticationInfo

3.4.1.16.3.12.1 - CSipHeader::GetAuthenticationInfo Method

Provides access to the Authentication-Info header's ainfo construct.

C++

```
const CGenericParam& GetAuthenticationInfo() const;
CGenericParam& GetAuthenticationInfo();
```

Returns

The value of the header, as a parameter (CGenericParam (see page 140)).

Description

Provides information for mutual authentication with HTTP Digest.

For example, in the header:

Authentication-Info: nextnonce="47364c23432d2e131a5fb210812c",nc=4

GetAuthenticationInfo on this CSipHeader (see page 239) will give the parameter nextnonce set to "47364c23432d2e131a5fb210812c".

GetAuthenticationInfo on the next CSipHeader (see page 239) will give the parameter nc set to 4.

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.13 - GetAuthorizationScheme

3.4.1.16.3.13.1 - CSipHeader::GetAuthorizationScheme Method

Provides access to the Authorization header's auth-scheme construct.

C++

```
const CToken& GetAuthorizationScheme() const;
CToken& GetAuthorizationScheme();
```

Returns

The token (CToken (see page 360)) in the Authorization header field.

Description

Contains the authentication credentials of a UA. For example, in the header:

```
Authorization: Digest username="Alice", realm="atlanta.com",
nonce="84a4cc6f3082121f32b42a2187831a9e",
response="7587245234b3434cc3412213e5f113a5432"
```

this is the token containing "Digest".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.14 - GetCallId

3.4.1.16.3.14.1 - CSipHeader::GetCallId Method

Provides access to the Call-Id header's callid construct.

C++

```
const CToken& GetCallId() const;
CToken& GetCallId();
```

Returns

The token(CToken (see page 360)) in the Call-ID header field.

Description

Uniquely identifies a particular invitation or all registrations of a particular client. For example, in the header:

```
Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com
```

this is the token containing "f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.15 - GetCallInfo

3.4.1.16.3.15.1 - CSipHeader::GetCallInfo Method

Provides access to the Call-Info header's absoluteURI construct.

C++

```
const CNameAddr& GetCallInfo() const;
CNameAddr& GetCallInfo();
```

Returns

The name address (CNameAddr (see page 201)) in the Call-Info header field.

Description

Provides additional information about the caller or callee, depending on whether it is found in a request or response. For example, in the header:

```
Call-Info: <http://www.example.com/alice/photo.jpg> ;purpose=icon,
<http://www.example.com/alice/> ;purpose=info
```

this is the name address containing the URI "http://www.example.com/alice/photo.jpg".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.16 - GetContact

3.4.1.16.3.16.1 - CSipHeader::GetContact Method

Provides access to the Contact header's (name-addr / addr-spec) construct.

C++

```
const CNameAddr& GetContact() const;
CNameAddr& GetContact();
```

Returns

The name address (CNameAddr (see page 201)) in the Contact header field.

Description

Provides a URI whose meaning depends on the type of request or response in which it is. It has a role similar to the Location header field in HTTP. For example, in the header:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
;q=0.7; expires=3600,
"Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
```

this is the name address containing the URI "sip:watson@worcester.bell-telephone.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.17 - GetContentDescription

3.4.1.16.3.17.1 - CSipHeader::GetContentDescription Method

Provides access to the Content-Description header's text construct.

C++

```
const CToken& GetContentDescription() const;
CToken& GetContentDescription();
```

Returns

The token (CToken (see page 360)) in the Content-Description header field.

Description

Provides the ability to associate some descriptive information with a given body. For example, in the header:

```
Content-Description: This is the content description
```

this is the token containing "This is the content description".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.18 - GetContentDisposition

3.4.1.16.3.18.1 - CSipHeader::GetContentDisposition Method

Provides access to the Content-Disposition header's disp-type construct.

C++

```
const CToken& GetContentDisposition() const;
CToken& GetContentDisposition();
```

Returns

The token (CToken (see page 360)) in the Content-Disposition header field.

Description

Describes how the message body or, for multipart messages, a message body part, is to be interpreted by the UAC or UAS. This SIP header field extends the MIME Content-Type (RFC 2183 [18]). For example, in the header:

```
Content-Disposition: session
```

this is the token containing "session".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.19 - GetContentEncoding**3.4.1.16.3.19.1 - CSipHeader::GetContentEncoding Method**

Provides access to the Content-Encoding header's content-coding construct.

C++

```
const CToken& GetContentEncoding() const;
CToken& GetContentEncoding();
```

Returns

The token (CToken (see page 360)) in the Content-Encoding header field.

Description

Used as a modifier to the "media-type". When present, its value indicates what additional content encodings have been applied to the entity-body, and thus what decoding mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header field. For example, in the header:

```
Content-Encoding: gzip
```

this is the token containing "gzip".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.20 - GetContentId**3.4.1.16.3.20.1 - CSipHeader::GetContentId Method**

Provides access to the Content-ID header's addr-spec (RFC 2822) construct.

C++

```
const CMailboxUri& GetContentId() const;
CMailboxUri& GetContentId();
```

Returns

The mailbox URI (CMailboxUri (see page 182)) in the Content-ID header field.

Description

Provides a URI whose meaning depends on the type of the body in which it is. For example, in the header.:

```
Content-ID: <foo@bar.net>
```

This is the address containing the URI "foo@bar.net".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.21 - GetContentLanguage**3.4.1.16.3.21.1 - CSipHeader::GetContentLanguage Method**

Provides access to the Content-Language header's language-tag construct.

C++

```
const CToken& GetContentLanguage() const;
CToken& GetContentLanguage();
```

Returns

The token (CToken (see page 360)) in the Content-Language header field.

Description

Describes the natural language(s) of the intended audience for the enclosed entity. For example, in the header:

Content-Language: fr

this is the token containing "fr".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.22 - GetContentLength**3.4.1.16.3.22.1 - CSipHeader::GetContentLength Method**

Provides access to the Content-Length header's 1*DIGIT construct.

C++

```
const CToken& GetContentLength() const;
CToken& GetContentLength();
```

Returns

The token (CToken (see page 360)) in the Content-Length header field.

Description

Indicates the size of the message-body, in decimal number of octets, sent to the recipient. For example, in the header:

Content-Length: 349

this is the token containing "349".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.23 - GetContentTransferEncoding**3.4.1.16.3.23.1 - CSipHeader::GetContentTransferEncoding Method**

Provides access to the Content-Language header's mechanism construct.

C++

```
const CToken& GetContentTransferEncoding() const;
CToken& GetContentTransferEncoding();
```

Returns

The token (CToken (see page 360)) in the Content-Transfer-Encoding header field.

Description

Specifies the encoding transformation that was applied to the body For example, in the header:

```
Content-Transfer-Encoding: gzip
```

this is the token containing "gzip".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.24 - GetContentTypeMSubType**3.4.1.16.3.24.1 - CSipHeader::GetContentTypeMSubType Method**

Provides access to the Content-Type header's m-subtype construct.

C++

```
const CToken& GetContentTypeMSubType() const;
CToken& GetContentTypeMSubType();
```

Returns

The second token (CToken (see page 360)) in the Content-Type header field.

Description

Indicates the media sub-type of the message-body sent to the recipient. For example, in the header:

```
Content-Type: application/sdp
```

this is the token containing "sdp".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.25 - GetContentTypeMType**3.4.1.16.3.25.1 - CSipHeader::GetContentTypeMType Method**

Provides access to the Content-Type header's m-type construct.

C++

```
const CToken& GetContentTypeMType() const;
CToken& GetContentTypeMType();
```

Returns

The first token (CToken (see page 360)) in the Content-Type header field.

Description

Indicates the media type of the message-body sent to the recipient. For example in the header:

```
Content-Type: application/sdp
```

this is the token containing "application".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.26 - GetCSeqMethod

3.4.1.16.3.26.1 - CSipHeader::GetCSeqMethod Method

Provides access to the CSeq header's Method construct.

C++

```
const CToken& GetCSeqMethod() const;
CToken& GetCSeqMethod();
```

Returns

The second token (CToken (see page 360)) in the CSeq header field.

Description

Contains the request method. For example, in the header:

```
CSeq: 4711 INVITE
```

this is the token containing "INVITE".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.27 - GetCSeqNumber

3.4.1.16.3.27.1 - CSipHeader::GetCSeqNumber Method

Provides access to the CSeq header's 1*DIGIT construct.

C++

```
const CToken& GetCSeqNumber() const;
CToken& GetCSeqNumber();
```

Returns

The first token (CToken (see page 360)) in the CSeq header field.

Description

Contains a single decimal sequence number. For example, in the header:

```
CSeq: 4711 INVITE
```

this is the token containing "4711".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.28 - GetDate

3.4.1.16.3.28.1 - CSipHeader::GetDate Method

Provides access to the Date header's SIP-date construct.

C++

```
const CDate& GetDate() const;
CDate& GetDate();
CNameAddr& GetErrorInfo();
```

Returns

The date (CDate (see page 133)) in the Date header field.

Description

Contains the date and time. For example, in the header:

Date: Sat, 13 Nov 2010 23:29:00 GMT

this is the date containing "Sat, 13 Nov 2010 23:29:00 GMT".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.29 - GetDiversion

3.4.1.16.3.29.1 - CSipHeader::GetDiversion Method **New in 4.1.4**

Provides access to the Diversion header's name-addr.

C++

```
const CNameAddr& GetDiversion() const;
CNameAddr& GetDiversion();
```

Returns

The name address(CNameAddr (see page 201)) in the Diversion header field.

Description

Provides an URI indicating from whom the call was diverted. For example in the header:

Diversion: <sip:Bob@uas1.isp.com>
;reason=do-not-disturb

this will be the name address containing the URI "sip:Bob@uas1.isp.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.30 - GetErrorInfo

3.4.1.16.3.30.1 - CSipHeader::GetErrorInfo Method

Provides access to the Error-Info header's error-uri construct.

C++

```
const CNameAddr& GetErrorInfo() const;
```

Returns

The name address (CNameAddr (see page 201)) in the Error-Info header field.

Description

Provides a pointer to additional information about the error status response. For example, in the header:

Error-Info: <sip:not-in-service-recording@atlanta.com>

this is the name address containing the URI "sip:not-in-service-recording@atlanta.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.31 - GetEvent

3.4.1.16.3.31.1 - CSipHeader::GetEvent Method

Provides access to the Event header's event-type construct.

C++

```
const CToken& GetEvent() const;
CToken& GetEvent();
```

Returns

The token (CToken (see page 360)) in the Event header field.

Description

Used to indicate which event or class of events the message contains or subscribes. For example, in the header:

```
Event: spirits-INDPs
```

this is the token containing "spirits-INDPs".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.32 - GetExpires**3.4.1.16.3.32.1 - CSipHeader::GetExpires Method**

Provides access to the Expires header's delta-seconds construct.

C++

```
const CToken& GetExpires() const;
CToken& GetExpires();
CToken& GetFlowTimer();
```

Returns

The token (CToken (see page 360)) in the Expires header field.

Description

Gives the relative time after which the message (or content) expires. The value of this field is an integral number of seconds (in decimal) between 0 and (2**32)-1, measured from the receipt of the request. For example, in the header:

```
Expires: 5
```

this is the token containing "5".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.33 - GetExtensionHeaderName**3.4.1.16.3.33.1 - CSipHeader::GetExtensionHeaderName Method**

Provides access to the Extension-header's header-name construct.

C++

```
const CToken& GetExtensionHeaderName() const;
CToken& GetExtensionHeaderName();
```

Returns

The token (CToken (see page 360)) containing the name of a generic header field.

Description

The name of an extension header. Extension headers are special since their name is not parsed by this class, but by the user of the CSipHeader (see page 239).

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.34 - GetExtensionHeaderValue**3.4.1.16.3.34.1 - CSipHeader::GetExtensionHeaderValue Method**

Provides access to the Extension-header's header-value construct.

C++

```
const CToken& GetExtensionHeaderValue() const;
CToken& GetExtensionHeaderValue();
```

Returns

The token (CToken (see page 360)) containing the value of a generic header field.

Description

The value of an extension header.

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.35 - GetFlowTimer**3.4.1.16.3.35.1 - CSipHeader::GetFlowTimer Method**

Provides access to the Flow Timer header's delta-seconds construct.

C++

```
const CToken& GetFlowTimer() const;
```

Returns

The token(CToken (see page 360)) in the Flow-Timer header field.

Description

Gives the number of seconds the server is prepared to wait without seeing keep alives before it considers the corresponding flow dead.
For example in the header:

Flow-timer: 120

this will be the token containing "120".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.36 - GetFrom**3.4.1.16.3.36.1 - CSipHeader::GetFrom Method**

Provides access to the From header's from-spec construct.

C++

```
const CNameAddr& GetFrom() const;
CNameAddr& GetFrom();
```

Returns

The name address (CNameAddr (see page 201)) in the From header field.

Description

Indicates the initiator of the request. This may be different from the initiator of the dialog. Requests sent by the callee to the caller use the callee's address in the From header field. For example, in the header:

```
From: "A. G. Bell" <sip:agb@bell-telephone.com> ;tag=a48s
```

this is the name address containing the URI "sip:agb@bell-telephone.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.37 - CSipHeader::GetHeaderName Method

Retrieves the long or short form following ShortNameForm configuration.

C++

```
const char* GetHeaderName() const;
```

Returns

Name of the header. It can be either short or long form. The short form is returned if the configuration of short form is enabled. If short form configuration is set, but no short form is found, the long name is returned. If no name is found, the return is NULL.

Description

Finds the header's name and returns it.

See Also

SetNameForm (see page 293)

3.4.1.16.3.38 - CSipHeader::GetHeaderType Method

Returns the header type.

C++

```
ESipHeaderType GetHeaderType() const;
```

Returns

The header's type.

Description

Returns the header's type.

3.4.1.16.3.39 - GetHistoryInfo

3.4.1.16.3.39.1 - CSipHeader::GetHistoryInfo Method

Provides access to the History-Info header's hi-targeted-to-uri construct.

C++

```
const CNameAddr& GetHistoryInfo() const;
CNameAddr& GetHistoryInfo();
```

Returns

The name address (CNameAddr (see page 201)) in the History-Info header field.

Description

Provides a standard mechanism for capturing the request history information. For example, in the header:

```
History-Info: <sip:UserA@examplenetwork.com?Reason=SIP;
cause=302; text="Moved Temporarily">; index=1.1,
```

```
<sip:UserB@examplenetwork.com? Reason=SIP;cause=486;
text="Busy Here">;index=1.2,
<sip:45432@vm.examplenetwork.com> ; index=1.3
```

this is the name address containing the URI "sip:UserA@examplenetwork.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.40 - GetIdentity

3.4.1.16.3.40.1 - CSipHeader::GetIdentity Method

Provides access to the Identity header's signed-identity-digest construct.

C++

```
const CToken& GetIdentity() const;
CToken& GetIdentity();
```

Returns

The token (CToken (see page 360)) in the Identity header field.

Description

Contains a signature using the domain's private key. For example, in the header:

```
Identity: "dKJ97..."
```

this is the token containing "dKJ97....".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetIdentityInfo (see page 261)

3.4.1.16.3.41 - GetIdentityInfo

3.4.1.16.3.41.1 - CSipHeader::GetIdentityInfo Method

Provides access to the Identity-Info header's ident-info construct.

C++

```
const CNameAddr& GetIdentityInfo() const;
CNameAddr& GetIdentityInfo();
```

Returns

The name address (CNameAddr (see page 201)) in the Identity-Info header field.

Description

Contains the corresponding certificate to the Identity header. For example, in the header:

```
Identity-Info: <https://example.com/cert>;alg=rsa-sha1
```

this is the name address containing the URI "https://example.com/cert".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetIdentity (see page 261)

3.4.1.16.3.42 - GetInReplyTo

3.4.1.16.3.42.1 - CSipHeader::GetInReplyTo Method

Provides access to the In-Reply-To header's callid construct.

C++

```
const CToken& GetInReplyTo() const;
CToken& GetInReplyTo();
```

Returns

The token (CToken (see page 360)) in the In-Reply-To header field.

Description

Enumerates the Call-IDs that this call references or returns. For example, in the header:

```
In-Reply-To: 70710@saturn.bell-tel.com, 17320@saturn.bell-tel.com
```

this is the token containing "70710@saturn.bell-tel.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.43 - GetJoin**3.4.1.16.3.43.1 - CSipHeader::GetJoin Method**

Provides access to the Join header's callid construct.

C++

```
const CToken& GetJoin() const;
CToken& GetJoin();
```

Returns

The token (CToken (see page 360)) in the Join header field.

Description

Indicates that a new dialog (created by the INVITE in which the Join header field is contained) should be joined with a dialog identified by the header field, and any associated dialogs or conferences. For example, in the header:

```
Join: 98732@sip.example.com
;from-tag=r33th4x0r
;to-tag=ff87ff
```

this is the token containing "98732@sip.example.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.44 - CSipHeader::GetLongHeaderName Method

Retrieves the long form header name for this header.

C++

```
const char* GetLongHeaderName() const;
```

Returns

Long form of the header name. It can be NULL if the type is eHDR_EXTENSION and the header's name is one character long or missing.

Description

Finds the header's long name form.

See Also

GetHeaderName (see page 260), GetShortHeaderName (see page 282), SetNameForm (see page 293)

3.4.1.16.3.45 - GetMaxForwards

3.4.1.16.3.45.1 - CSipHeader::GetMaxForwards Method

Provides access to the Max-Forwards header's 1*DIGIT construct.

C++

```
const CToken& GetMaxForwards() const;
CToken& GetMaxForwards();
```

Returns

The token (CToken (see page 360)) in the Max-Forwards header field.

Description

Limits the number of proxies or gateways that can forward the request to the next downstream server. For example, in the header:

Max-Forwards: 6

this is the token containing "6".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.46 - GetMimeType

3.4.1.16.3.46.1 - CSipHeader::GetMimeType Method

Provides access to the MIME-Version header's "1*DIGIT ." 1*DIGIT" construct.

C++

```
const CToken& GetMimeType() const;
CToken& GetMimeType();
```

Returns

The token (CToken (see page 360)) in the MIME-Version header field.

Description

Indicates the version of the MIME protocol used to construct the message. For example, in the header:

MIME-Version: 1.0

this is the token containing "1.0".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.47 - GetMinExpires

3.4.1.16.3.47.1 - CSipHeader::GetMinExpires Method

Provides access to the Min-Expires header's delta-seconds construct.

C++

```
const CToken& GetMinExpires() const;
CToken& GetMinExpires();
```

Returns

The token (CToken (see page 360)) in the Min-Expires header field.

Description

Conveys the minimum refresh interval supported for soft-state elements managed by that server. For example, in the header:

Min-Expires: 60

this is the token containing "60".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.48 - GetMinSe**3.4.1.16.3.48.1 - CSipHeader::GetMinSe Method**

Provides access to the Min-SE header's delta-seconds construct.

C++

```
const CToken& GetMinSe() const;
CToken& GetMinSe();
```

Returns

The token (CToken (see page 360)) in the Min-SE header field.

Description

Indicates the minimum value for the session interval in seconds. For example, in the header:

Min-SE: 210

this is the token containing "210".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetSessionExpires (see page 281)

3.4.1.16.3.49 - CSipHeader::GetNbNextHeaders Method

Returns the number of headers in the chain, but omits the current header.

C++

```
unsigned int GetNbNextHeaders() const;
```

Returns

Number of headers in the chain, excluding the present instance.

Description

This method counts the number of CSipHeader (see page 239) pointers that are linked in the chain. Note that a raw CSipHeader (see page 239) may hide multiple headers if these are comma-separated, and that this method would count that raw header as only one header, until it is parsed.

See Also

GetNbParsedHeaders (see page 264)

3.4.1.16.3.50 - CSipHeader::GetNbParsedHeaders Method

Returns the number of parsed headers in the chain, including the current header.

C++

```
unsigned int GetNbParsedHeaders() const;
```

Returns

Number of parsed headers in the chain, including the present instance.

Description

This method browses through the chain and counts the headers that are parsed. The current instance is included in that count.

See Also

GetNbNextHeaders (see page 264)

3.4.1.16.3.51 - GetNextHeader**3.4.1.16.3.51.1 - CSipHeader::GetNextHeader Method**

Returns a pointer to the chained header at the given 0-based index.

C++

```
inline const CSipHeader* GetNextHeader(IN unsigned int uIndex = 0) const;
inline CSipHeader* GetNextHeader(IN unsigned int uIndex = 0);
```

Parameters

Parameters	Description
IN unsigned int uIndex = 0	0-based index of the header to find in the chain.
IN unsigned int uIndex = 0	0-based index of the header to find in the chain.

Returns

Pointer to the header at the specified index, or NULL if out of range.

Description

Finds the header at the specified index in the chain and returns it. This method accesses the list of next headers, which does not include the current instance of CSipHeader (see page 239).

See Also

AppendNextHeader (see page 246)

3.4.1.16.3.52 - GetOrganization**3.4.1.16.3.52.1 - CSipHeader::GetOrganization Method**

Provides access to the Organization header's [TEXT-UTF8-TRIM] construct.

C++

```
const CToken& GetOrganization() const;
CToken& GetOrganization();
```

Returns

The token (CToken (see page 360)) in the Organization header field.

Description

Conveys the name of the organization to which the SIP element issuing the request or response belongs. For example, in the header:

Organization: Boxes by Bob

this is the token containing "Boxes by Bob".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.53 - GetPAccessNetworkInfo

3.4.1.16.3.53.1 - CSipHeader::GetPAccessNetworkInfo Method

Provides access to the P-Access-Network-Info header's access-type construct.

C++

```
const CToken& GetPAccessNetworkInfo() const;
CToken& GetPAccessNetworkInfo();
```

Returns

The token (CToken (see page 360)) in the P-Access-Network-Info header field.

Description

The P-Access-Network-Info header is useful in SIP-based networks that also provide layer 2/layer 3 connectivity through different access technologies. SIP User Agents may use this header to relay information about the access technology to proxies that are providing services. For example, in the header:

```
P-Access-Network-Info: 3GPP-UTRAN-TDD; utran-cell-id-3gpp=C359A3913B20E
```

this is the token containing "3GPP-UTRAN-TDD".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.54 - GetParam

3.4.1.16.3.54.1 - CSipHeader::GetParam Method

Gets a header parameter value from the parameter list.

C++

```
inline const CToken* GetParam(IN const char* szName) const;
```

Parameters

Parameters	Description
IN const char* szName	Name of the parameter for which to look. A pre-defined list of parameters exists in SipParser/HeaderParameter.h.

Returns

A pointer to the value of the requested parameter. It is NULL if it did not exist.

Description

Returns a header parameter value or NULL if the parameter name is not found in the list.

See Also

GetParamList (see page 267), SetParamList (see page 294)

3.4.1.16.3.54.2 - CSipHeader::GetParam Method

Gets a header parameter value from the parameter list.

C++

```
CToken* GetParam(IN const char* szName, IN const EParameterCreationBehavior eParameterCreationBehavior =
ePARAM_DONT_CREATE, INOUT mxT_result* pRes = NULL);
```

Parameters

Parameters	Description
IN const char* szName	Name of the parameter for which to look. A pre-defined list of parameters exists in SipParser/HeaderParameter.h.
IN const EParameterCreationBehavior eParameterCreationBehavior = ePARAM_DONT_CREATE	Enum that specifies the behaviour of this method: <ul style="list-style-type: none"> • ePARAM_DONT_CREATE: A NULL pointer is returned when the szName is not found from the parameter list. • ePARAM_CREATE_NEW: Adds a new parameter to the list when the szName is not found from the parameter list.
INOUT mxt_result* pRes = NULL	Result, it can be NULL.
resS_OK	Success, parameter value is returned.
resFE_UNEXPECTED	This header cannot have parameters as per the header definition table.

Returns

A pointer to the value of the requested parameter. It is NULL if it did not exist and ePARAM_DONT_CREATE was specified.

Description

Returns a header parameter value. It can return a NULL when eParameterCreationBehavior is set to ePARAM_DONT_CREATE and the parameter name is not found in the list.

See Also

GetParamList (see page 267), SetParamList (see page 294)

3.4.1.16.3.55 - GetParamList

3.4.1.16.3.55.1 - CSipHeader::GetParamList Method

Provides access to the parameter list.

C++

```
inline const CGenParamList* GetParamList(INOUT mxt_result* pres = NULL) const;
inline CGenParamList* GetParamList(INOUT mxt_result* pres = NULL);
```

3.4.1.16.3.55.2 - CSipHeader::GetParamList Method

Provides access to the parameter list.

C++

```
inline const CGenParamList* GetParamList(OUT mxt_result& rres) const;
inline CGenParamList* GetParamList(OUT mxt_result& rres);
```

Parameters

Parameters	Description
OUT mxt_result& rres	Reference to a mxt_result where the result is set. resS_OK Pointer set in the OUT parameter. resFE_UNEXPECTED This header cannot have parameters as per the header definition table. resSW_SIPPARSER_NULL_PTR Warning, the value is NULL.

Returns

Pointer to the internal parameter list. It can be NULL

Description

Provides access to the optional parameter list.

See Also

SetParamList (see page 294)

3.4.1.16.3.56 - GetPAssertedIdentity

3.4.1.16.3.56.1 - CSipHeader::GetPAssertedIdentity Method

Provides access to the P-Asserted-Identity header's PAssertedID-value construct.

C++

```
const CNameAddr& GetPAssertedIdentity() const;
CNameAddr& GetPAssertedIdentity();
```

Returns

The name address (CNameAddr (see page 201)) in the P-Asserted-Identity header field.

Description

The P-Asserted-Identity header field is used among trusted SIP entities (typically intermediaries) to carry the identity of the user sending a SIP message as it was verified by authentication. For example, in the header:

```
P-Asserted-Identity: "Cullen Jennings" <sip:fluffy@domain.com>
```

this is the name address containing the URI "sip:fluffy@domain.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetPPreferredIdentity (see page 270)

3.4.1.16.3.57 - GetPAssociatedUri

3.4.1.16.3.57.1 - CSipHeader::GetPAssociatedUri Method

Provides access to the P-Associated-URI header's name-addr construct.

C++

```
const CNameAddr& GetPAssociatedUri() const;
CNameAddr& GetPAssociatedUri();
```

Returns

The name address (CNameAddr (see page 201)) in the P-Associated-URI header field.

Description

Allows a registrar to return a set of associated URIs for a registered address-of-record. For example, in the header:

```
P-Associated-URI: <sip:alice-family@home1.net>,
<sip:alice-business@home1.net>,
<sip:+46-8-123-4567>@home1.net;user=phone>
```

this is the name address containing the URI "sip:alice-family@home1.net".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.58 - GetPath

3.4.1.16.3.58.1 - CSipHeader::GetPath Method

Provides access to the Path header's name-addr construct.

C++

```
const CNameAddr& GetPath() const;
CNameAddr& GetPath();
```

Returns

The name address (CNameAddr (see page 201)) in the Path header field.

Description

The Path header is a SIP extension header field with a syntax very similar to the Record-Route header field. It is used in conjunction with SIP REGISTER requests and 200 class messages in response to REGISTER (REGISTER responses). For example, in the header:

```
Path: <sip:P3.EXAMPLEHOME.COM;lr>,<sip:P1.EXAMPLEVISITED.COM;lr>
```

this is the name address containing the URI "sip:P3.EXAMPLEHOME.COM".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.59 - GetPCalledPartyId**3.4.1.16.3.59.1 - CSipHeader::GetPCalledPartyId Method**

Provides access to the P-Called-Party-ID header's name-addr construct.

C++

```
const CNameAddr& GetPCalledPartyId() const;
CNameAddr& GetPCalledPartyId();
```

Returns

The name address (CNameAddr (see page 201)) in the P-Called-Party-ID header field.

Description

Identifies the address-of-record to which this session is destined. For example, in the header:

```
P-Called-Party-ID: sip:user1-business@example.com
```

this is the name address containing the URI "sip:user1-business@example.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.60 - GetPChargingFunctionAddresses**3.4.1.16.3.60.1 - CSipHeader::GetPChargingFunctionAddresses Method**

Provides access to the P-Charging-Function-Addresses header's mandatory first charge-addr-params construct. Other charge-addr-params are accessible through the parameter list.

C++

```
const CGenericParam& GetPChargingFunctionAddresses() const;
CGenericParam& GetPChargingFunctionAddresses();
```

Returns

The first parameter (CGenericParam (see page 140)) in the P-Charging-Function-Addresses header field.

Description

Contains one or more parameter that contains the hostnames or IP addresses of the nodes willing to receive charging information. For example, in the header:

```
P-Charging-Function-Addresses: ccf=[5555::b99:c88:d77:e66];
ccf=[5555::a55:b44:c33:d22]; ecf=[5555::1ff:2ee:3dd:4cc];
ecf=[5555::6aa:7bb:8cc:9dd]
```

this is the parameter ccf set to "[5555::b99:c88:d77:e66]"

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.61 - GetPChargingVector**3.4.1.16.3.61.1 - CSipHeader::GetPChargingVector Method**

Provides access to the P-Charging-Vector header's icid-value construct.

C++

```
const CGenericParam& GetPChargingVector() const;
CGenericParam& GetPChargingVector();
```

Returns

The first parameter (CGenericParam (see page 140)) in the P-Charging-Vector header field.

Description

Conveys charging related information, such as the globally unique IMS charging identifier (ICID) value. For example, in the header:

```
P-Charging-Vector: icid-value="AyretyU0dm+6O2IrT5tAFrbHLso=023551024";
orig-roi=homel.net; term-roi=home2.net
```

this is the parameter icid-value set to "AyretyU0dm+6O2IrT5tAFrbHLso=023551024".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.62 - GetPMediaAuthorization**3.4.1.16.3.62.1 - CSipHeader::GetPMediaAuthorization Method**

Provides access to the P-Media-Authorization header's P-Media-Authorization-Token construct.

C++

```
const CToken& GetPMediaAuthorization() const;
CToken& GetPMediaAuthorization();
```

Returns

The token (CToken (see page 360)) in the P-Media-Authorization header field.

Description

Contains one or more media authorization tokens to be included in subsequent resource reservations for the media flows associated with the session, that is, passed to an independent resource reservation mechanism, which is not specified here. For example, in the header:

```
P-Media-Authorization: 123456789abcdef
```

this is the token containing "123456789abcdef".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.63 - GetPPreferredIdentity**3.4.1.16.3.63.1 - CSipHeader::GetPPreferredIdentity Method**

Provides access to the P-Preferred-Identity header's PPreferredID-value construct.

C++

```
const CNameAddr& GetPPREFERREDIdentity() const;
CNameAddr& GetPPREFERREDIdentity();
```

Returns

The name address (CNameAddr (see page 201)) in the P-Preferred-Identity header field.

Description

The P-Preferred-Identity header field is used from a user agent to a trusted proxy to carry the identity that the user sending the SIP message wants to be used as the P-Asserted-Header field value, which the trusted element inserts. For example, in the header:

```
P-Preferred-Identity: "Cullen Jennings" <sip:fluffy@domain.com>
```

this is the name address containing the URI "sip:fluffy@domain.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetPAssertedIdentity (see page 268)

3.4.1.16.3.64 - GetPriority

3.4.1.16.3.64.1 - CSipHeader::GetPriority Method

Provides access to the Priority header's priority-value construct.

C++

```
const CToken& GetPriority() const;
CToken& GetPriority();
```

Returns

The token (CToken (see page 360)) in the Priority header field.

Description

Indicates the urgency of the request as perceived by the client. For example, in the header:

```
Priority: emergency
```

this is the token containing "emergency".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.65 - GetPrivacy

3.4.1.16.3.65.1 - CSipHeader::GetPrivacy Method

Provides access to the Privacy header's priv-value construct.

C++

```
const CToken& GetPrivacy() const;
CToken& GetPrivacy();
```

Returns

The token (CToken (see page 360)) in the Privacy header field.

Description

Used by User-Agents to request privacy services from the network. For example, in the header:

```
Privacy: none
```

this is the token containing "none".

See Also

Header accessor nomenclature in CSipHeader ([see page 239](#)).

3.4.1.16.3.66 - GetProxyAuthenticateScheme**3.4.1.16.3.66.1 - CSipHeader::GetProxyAuthenticateScheme Method**

Provides access to the Proxy-Authenticate header's auth-scheme construct.

C++

```
const CToken& GetProxyAuthenticateScheme() const;
CToken& GetProxyAuthenticateScheme();
```

Returns

The token (CToken ([see page 360](#))) in the Proxy-Authenticate header field.

Description

Contains an authentication challenge. For example, in the header:

```
Proxy-Authenticate: Digest realm="atlanta.com",
domain="sip:ssl.carrier.com", qop="auth",
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5
```

this is the token containing "Digest".

See Also

Header accessor nomenclature in CSipHeader ([see page 239](#)). GetProxyAuthorizationScheme ([see page 272](#))

3.4.1.16.3.67 - GetProxyAuthorizationScheme**3.4.1.16.3.67.1 - CSipHeader::GetProxyAuthorizationScheme Method**

Provides access to the Proxy-Authorization header's auth-scheme construct.

C++

```
const CToken& GetProxyAuthorizationScheme() const;
CToken& GetProxyAuthorizationScheme();
```

Returns

The token (CToken ([see page 360](#))) in the Proxy-Authorization header field.

Description

Allows the client to identify itself (or its user) to a proxy that requires authentication. A Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested. For example, in the header:

```
Proxy-Authorization: Digest username="Alice", realm="atlanta.com",
nonce="c60f3082ee1212b402a21831ae",
response="245f23415f11432b3434341c022"
```

this is the token containing "Digest".

See Also

Header accessor nomenclature in CSipHeader ([see page 239](#)). GetProxyAuthenticateScheme ([see page 272](#))

3.4.1.16.3.68 - GetProxyRequire

3.4.1.16.3.68.1 - CSipHeader::GetProxyRequire Method

Provides access to the Proxy-Require header's option-tag construct.

C++

```
const CToken& GetProxyRequire() const;
CToken& GetProxyRequire();
```

Returns

The token (CToken (see page 360)) in the Proxy-Require header field.

Description

Indicates proxy-sensitive features that must be supported by the proxy. For example, in the header:

Proxy-Require: foo

this is the token containing "foo".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.69 - GetPVisitedNetworkId

3.4.1.16.3.69.1 - CSipHeader::GetPVisitedNetworkId Method

Provides access to the P-Visited-Network-ID header's (token / quoted-string) construct.

C++

```
const CToken& GetPVisitedNetworkId() const;
CToken& GetPVisitedNetworkId();
```

Returns

The token (CToken (see page 360)) in the P-Visited-Network-ID header field.

Description

Conveys to the registrar or home proxy in the home network the identifier of a visited network. The identifier is a text string or token that is known by both the registrar or the home proxy at the home network and the proxies in the visited network. For example, in the header:

P-Visited-Network-ID: "Visited network number 1"

this is the token containing "Visited network number 1".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.70 - GetRackCSeq

3.4.1.16.3.70.1 - CSipHeader::GetRackCSeq Method

Provides access to the RAck header's CSeq-num construct.

C++

```
const CToken& GetRackCSeq() const;
CToken& GetRackCSeq();
```

Returns

The second token (CToken (see page 360)) in the RAck header field.

Description

Indicates the value of the CSeq in the response that is being acknowledged. For example, in the header:

```
RAck: 776655 1 INVITE
```

this is the token containing "1".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.71 - GetRackMethod

3.4.1.16.3.71.1 - CSipHeader::GetRackMethod Method

Provides access to the RAck header's Method construct.

C++

```
const CToken& GetRackMethod() const;
CToken& GetRackMethod();
```

Returns

The third token (CToken (see page 360)) in the RAck header field.

Description

Indicates the value of the method in the response that is being acknowledged. For example, in the header:

```
RAck: 776655 1 INVITE
```

this is the token containing "INVITE".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.72 - GetRackResponseNum

3.4.1.16.3.72.1 - CSipHeader::GetRackResponseNum Method

Provides access to the RAck header's response-num construct.

C++

```
const CToken& GetRackResponseNum() const;
CToken& GetRackResponseNum();
```

Returns

The first token (CToken (see page 360)) in the RAck header field.

Description

Indicates the value from the RSeq header in the provisional response that is being acknowledged. For example, in the header:

```
RAck: 776655 1 INVITE
```

this is the token containing "776655".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.73 - GetRawHeader

3.4.1.16.3.73.1 - CSipHeader::GetRawHeader Method

Provides access to the raw header.

C++

```
inline CRawHeader* GetRawHeader();
```

Returns

Pointer to the raw header, or NULL if unavailable.

Description

Provides access to the raw header.

See Also

SetRawHeader (see page 294)

3.4.1.16.3.73.2 - CSipHeader::GetRawHeader Method

Provides access to the raw header.

C++

```
inline const CRawHeader* GetRawHeader() const;
```

Returns

Pointer to the raw header, or NULL if unavailable.

Description

Provides access to the raw header.

See Also

SetRawHeader (see page 294)

3.4.1.16.3.74 - GetReason**3.4.1.16.3.74.1 - CSipHeader::GetReason Method**

Provides access to the Reason header's protocol construct.

C++

```
const CToken& GetReason() const;
CToken& GetReason();
```

Returns

The token (CToken (see page 360)) in the Reason header field.

Description

Indicates why a SIP request has been issued or why a provisional response has been sent. For example, in the header:

Reason: preemption ;cause=1 ;text="UA Preemption"

this is the token containing "preemption".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.75 - GetRecordRoute**3.4.1.16.3.75.1 - CSipHeader::GetRecordRoute Method**

Provides access to the Record-Route header's name-addr construct.

C++

```
const CNameAddr& GetRecordRoute() const;
CNameAddr& GetRecordRoute();
```

Returns

The name address(CNameAddr (see page 201)) in the Record-Route header field.

Description

The Record-Route header field is inserted by proxies in a request to force future requests in the dialog to be routed through the proxy. For example, in the header:

```
Record-Route: <sip:server10.biloxi.com;lr>,
               <sip:bigbox3.site3.atlanta.com;lr>
```

this is the name address containing the URI "sip:server10.biloxi.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.76 - GetReferredBy**3.4.1.16.3.76.1 - CSipHeader::GetReferredBy Method**

Provides access to the Referred-By header's referrer-uri construct.

C++

```
const CNameAddr& GetReferredBy() const;
CNameAddr& GetReferredBy();
```

Returns

The name address (CNameAddr (see page 201)) in the Referred-By header field.

Description

Carries a SIP URI representing the identity of thereferrer and, optionally, the Content-ID of a body part that provides a more secure statement of that identity. For example, in the header:

```
Referred-By: <sip:referrer@referrer.example>
               ;cid="20398823.2UWQFN309shb3@referrer.example"
```

this is the name address containing the URI "sip:referrer@referrer.example".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.77 - GetReferTo**3.4.1.16.3.77.1 - CSipHeader::GetReferTo Method**

Provides access to the Refer-To header's a(name-addr / addr-spec) construct.

C++

```
const CNameAddr& GetReferTo() const;
CNameAddr& GetReferTo();
```

Returns

The name address (CNameAddr (see page 201)) in the Refer-To header field.

Description

Provides a URI to reference. For example, in the header:

```
Refer-To: <sip:refertarget@target.example>
```

this is the name address containing the URI "sip:refertarget@target.example".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.78 - GetRejectContact

3.4.1.16.3.78.1 - CSipHeader::GetRejectContact Method

Provides access to the Reject-Contact header's "*" construct.

C++

```
const CToken& GetRejectContact() const;
CToken& GetRejectContact();
```

Returns

The token (CToken (see page 360)) in the Reject-Contact header field.

Description

Contains feature sets which, if matched by a UA, imply that the request should not be routed to that UA. For example, in the header:

```
Reject-Contact: *;actor="msg-taker";video
```

this is the token containing "*".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.79 - GetReplaces

3.4.1.16.3.79.1 - CSipHeader::GetReplaces Method

Provides access to the Replaces header's callid construct.

C++

```
const CToken& GetReplaces() const;
CToken& GetReplaces();
```

Returns

The token (CToken (see page 360)) in the Replaces header field.

Description

Indicates that a single dialog identified by the header field is to be shut down and logically replaced by the incoming INVITE in which it is contained. For example, in the header:

```
Replaces: 98732@sip.example.com
;from-tag=r33th4x0r
;to-tag=ff87ff
```

this is the token containing "98732@sip.example.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.80 - GetReplyTo

3.4.1.16.3.80.1 - CSipHeader::GetReplyTo Method

Provides access to the Reply-To header's (name-addr / addr-spec) construct.

C++

```
const CNameAddr& GetReplyTo() const;
CNameAddr& GetReplyTo();
```

Returns

The name address (CNameAddr (see page 201)) in the Reply-To header field.

Description

Contains a logical return URI that may be different from the From header field. For example, in the header:

```
Reply-To: Bob <sip:bob@biloxi.com>
```

this is the name address containing the URI "sip:bob@biloxi.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.81 - GetRequestDisposition

3.4.1.16.3.81.1 - CSipHeader::GetRequestDisposition Method

Provides access to the Request-Disposition header's directive construct.

C++

```
const CToken& GetRequestDisposition() const;
CToken& GetRequestDisposition();
```

Returns

The token (CToken (see page 360)) in the Request-Disposition header field.

Description

Specifies caller preferences for how a server should process a request. Its value is a list of tokens, each of which specifies a particular processing directive. For example, in the header:

```
Request-Disposition: proxy, recurse, parallel
```

this is the token containing "proxy".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.82 - GetRequire

3.4.1.16.3.82.1 - CSipHeader::GetRequire Method

Provides access to the Require header's option-tag construct.

C++

```
const CToken& GetRequire() const;
CToken& GetRequire();
```

Returns

The token (CToken (see page 360)) in the Require header field.

Description

Tells UASs about options that the UAC expects the UAS to support in order to process the request. For example, in the header:

```
Require: 100rel
```

this is the token containing "100rel".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.83 - GetResourcePriority

3.4.1.16.3.83.1 - CSipHeader::GetResourcePriority Method

Provides access to the Resource-Priority header's Resource-value construct.

C++

```
const CToken& GetResourcePriority() const;
CToken& GetResourcePriority();
```

Returns

The token (CToken (see page 360)) in the Resource-Priority header field.

Description

Marks a SIP request as wanting prioritized access to resources. For example, in the header:

```
Resource-Priority: dsn.flash, wps.3
```

this is the token containing "dsn.flash".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.84 - GetRetryAfter

3.4.1.16.3.84.1 - CSipHeader::GetRetryAfter Method

Provides access to the Retry-After header's delta-seconds construct.

C++

```
const CToken& GetRetryAfter() const;
CToken& GetRetryAfter();
```

Returns

The first token (CToken (see page 360)) in the Retry-After header field.

Description

This header can be used with a 500 (Server Internal Error) or 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. It can also be used with a 404 (Not Found), 413 (Request Entity Too Large), 480 (Temporarily Unavailable), 486 (Busy Here), 600 (Busy), or 603 (Decline) response to indicate when the called party anticipates being available again. The value of this field is a positive integer number of seconds (in decimal) after the time of the response. An optional "duration" parameter indicates how long the called party can be reached, starting at the initial time of availability. For example, in the header:

```
Retry-After: 18000;duration=3600
```

this is the token containing "18000".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.85 - GetRetryAfterComment

3.4.1.16.3.85.1 - CSipHeader::GetRetryAfterComment Method

Provides access to the Retry-After header's comment construct.

C++

```
const CToken& GetRetryAfterComment() const;
CToken& GetRetryAfterComment();
```

Returns

The second token(CToken (see page 360)) in the Retry-After header field.

Description

The comment in a Retry-After header. For example in the header:

```
Retry-After: 120 (I'm in a meeting)
```

this will be the token containing "I'm in a meeting".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.86 - GetRoute

3.4.1.16.3.86.1 - CSipHeader::GetRoute Method

Provides access to the Route header's name-addr construct.

C++

```
const CNameAddr& GetRoute() const;
CNameAddr& GetRoute();
```

Returns

The name address (CNameAddr (see page 201)) in the Route header field.

Description

Forces routing for a request through the listed set of proxies. For example, in the header:

```
Route: <sip:bigbox3.site3.atlanta.com;lr>,
<sip:server10.biloxi.com;lr>
```

this is the name address containing the URI "sip:bigbox3.site3.atlanta.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.87 - GetRSeq

3.4.1.16.3.87.1 - CSipHeader::GetRSeq Method

Provides access to the RSeq header's response-num construct.

C++

```
const CToken& GetRSeq() const;
CToken& GetRSeq();
```

Returns

The token (CToken (see page 360)) in the RSeq header field.

Description

Identifies provisional responses within a transaction. For example, in the header:

```
CSeq: 4711
```

this is the token containing "4711".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.88 - GetServer

3.4.1.16.3.88.1 - CSipHeader::GetServer Method

Provides access to the Server header's server-val construct.

C++

```
const CToken& GetServer() const;
CToken& GetServer();
```

Returns

The token (CToken (see page 360)) in the Server header field.

Description

Contains information about the software used by the UAS to handle the request. For example, in the header:

```
Server: HomeServer v2
```

this is the token containing "HomeServer v2".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.89 - GetServiceRoute

3.4.1.16.3.89.1 - CSipHeader::GetServiceRoute Method

Provides access to the Service-Route header's name-addr construct.

C++

```
const CNameAddr& GetServiceRoute() const;
CNameAddr& GetServiceRoute();
```

Returns

The name address (CNameAddr (see page 201)) in the Service-Route header field.

Description

Contains a route vector that directs requests through a specific sequence of proxies. For example, in the header:

```
Service-Route: <sip:P2.HOME.EXAMPLE.COM;lr>,
               <sip:HSP.HOME.EXAMPLE.COM;lr>
```

this is the name address containing the URI "sip:P2.HOME.EXAMPLE.COM".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.90 - GetSessionExpires

3.4.1.16.3.90.1 - CSipHeader::GetSessionExpires Method

Provides access to the Session-Expires header's delta-seconds construct.

C++

```
const CToken& GetSessionExpires() const;
CToken& GetSessionExpires();
```

Returns

The token (CToken (see page 360)) in the Session-Expires header field.

Description

Conveys the session interval for a SIP session. For example, in the header:

Session-Expires: 50

this is the token containing "50".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetMinSe (see page 264)

3.4.1.16.3.91 - CSipHeader::GetShortHeaderName Method

Retrieves the short form header name for this header, if available.

C++

```
const char* GetShortHeaderName() const;
```

Returns

Short form of the header name or NULL if the header type does not have a short form, or if the the header type is eHDR_EXTENSION and the name's length is more than one character long.

Description

Finds the header's short name form.

See Also

GetHeaderName (see page 260)

3.4.1.16.3.92 - GetSipETag**3.4.1.16.3.92.1 - CSipHeader::GetSipETag Method**

Provides access to the SIP-ETag header's entity-tag construct.

C++

```
const CToken& GetSipETag() const;
CToken& GetSipETag();
```

Returns

The token (CToken (see page 360)) in the SIP-ETag header field.

Description

The SIP-ETag header is present in the response sent by Event State Compositor (ESC) and contains a single entity-tag identifying the publication. For example, in the header:

SIP-ETag: dx200xyz

this is the token containing "dx200xyz".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetSipIfMatch (see page 283)

3.4.1.16.3.93 - GetSipIfMatch

3.4.1.16.3.93.1 - CSipHeader::GetSipIfMatch Method

Provides access to the SIP-If-Match header's entity-tag construct.

C++

```
const CToken& GetSipIfMatch() const;
CToken& GetSipIfMatch();
```

Returns

The token (CToken) (see page 360) in the SIP-If-Match header field.

Description

Identifies the specific event state that the request is refreshing, modifying, or removing. This header field MUST contain a single entity-tag that was returned by the ESC in the SIP-ETag header field of the response to a previous publication. For example, in the header:

SIP-IF-Match: dx200xyz

this is the token containing "dx200xyz".

See Also

Header accessor nomenclature in CSipHeader (see page 239). GetSipETag (see page 282)

3.4.1.16.3.94 - GetSubject

3.4.1.16.3.94.1 - CSipHeader::GetSubject Method

Provides access to the Subject header's [TEXT-UTF8-TRIM] construct.

C++

```
const CToken& GetSubject() const;
CToken& GetSubject();
```

Returns

The token (CToken) (see page 360) in the Subject header field.

Description

Provides a summary or indicates the nature of the call, allowing call filtering without having to parse the session description. For example, in the header:

Subject: Need more boxes

this is the token containing "Need more boxes".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.95 - GetSubscriptionState

3.4.1.16.3.95.1 - CSipHeader::GetSubscriptionState Method

Provides access to the Subscription-State header's substate-value construct.

C++

```
const CToken& GetSubscriptionState() const;
CToken& GetSubscriptionState();
```

Returns

The token (CToken) (see page 360) in the Subscription-State header field.

Description

Indicates in which state a subscription is. For example, in the header:

```
Subscription-State: active;expires=(depends on Refer-To URI)
```

this is the token containing "active".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.96 - GetSupported

3.4.1.16.3.96.1 - CSipHeader::GetSupported Method

Provides access to the Supported header's [option-tag] construct.

C++

```
const CToken& GetSupported() const;
CToken& GetSupported();
```

Returns

The token (CToken (see page 360)) in the Supported header field.

Description

Enumerates all the extensions supported by the UAC or UAS. For example, in the header:

```
Supported: 100rel
```

this is the token containing "100rel".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.97 - GetTargetDialog

3.4.1.16.3.97.1 - CSipHeader::GetTargetDialog Method

Provides access to the Target-Dialog header's callid construct.

C++

```
const CToken& GetTargetDialog() const;
CToken& GetTargetDialog();
```

Returns

The callid construct in the Target-Dialog header field.

Description

Provides access to the Target-Dialog header's callid construct.

For example, with the following Target-Dialog header:

```
Target-Dialog: jazz@festival.com
:local-tag=kkaz
:remote-tag=6544
```

the returned token contains "jazz@festival.com".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.98 - GetTimestamp

3.4.1.16.3.98.1 - CSipHeader::GetTimestamp Method

Provides access to the Timestamp header's 1*(DIGIT) ["." *(DIGIT)] [LWS delay] construct.

C++

```
const CToken& GetTimestamp() const;
CToken& GetTimestamp();
```

Returns

The token (CToken (see page 360)) in the Timestamp header field.

Description

Describes when the UAC sent the request to the UAS. For example, in the header:

Timestamp: 54

this is the token containing "54".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.99 - GetTo

3.4.1.16.3.99.1 - CSipHeader::GetTo Method

Provides access to the To header's (name-addr / addr-spec) construct.

C++

```
const CNameAddr& GetTo() const;
CNameAddr& GetTo();
```

Returns

The name address (CNameAddr (see page 201)) in the To header field.

Description

Specifies the logical recipient of the request. For example, in the header:

To: The Operator <sip:operator@cs.columbia.edu>;tag=287447

this is the name address containing the URI "sip:operator@cs.columbia.edu".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.100 - GetUnsupported

3.4.1.16.3.100.1 - CSipHeader::GetUnsupported Method

Provides access to the Unsupported header's option-tag construct.

C++

```
const CToken& GetUnsupported() const;
CToken& GetUnsupported();
```

Returns

The token (CToken (see page 360)) in the Unsupported header field.

Description

Lists the features not supported by the UAs. For example, in the header:

```
Unsupported: foo
```

this is the token containing "foo".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.101 - GetUserAgent

3.4.1.16.3.101.1 - CSipHeader::GetUserAgent Method

Provides access to the User-Agent header's server-val construct.

C++

```
const CToken& GetUserAgent() const;  
CToken& GetUserAgent();
```

Returns

The token (CToken (see page 360)) in the User-Agent header field.

Description

Contains information about the UAC originating the request. For example, in the header:

```
User-Agent: Softphone Beta1.5
```

this is the token containing "Softphone Beta1.5".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.102 - GetViaProtocolName

3.4.1.16.3.102.1 - CSipHeader::GetViaProtocolName Method

Provides access to the Via header's protocol-name construct.

C++

```
const CToken& GetViaProtocolName() const;  
CToken& GetViaProtocolName();
```

Returns

The first token (CToken (see page 360)) in the Via header field.

Description

Contains the name of the protocol used to send the message. For example, in the header:

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7
```

this is the token containing "SIP".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.103 - GetViaProtocolVersion

3.4.1.16.3.103.1 - CSipHeader::GetViaProtocolVersion Method

Provides access to the Via header's protocol-version construct.

C++

```
const CToken& GetViaProtocolVersion() const;
CToken& GetViaProtocolVersion();
```

Returns

The second token (CToken (see page 360)) in the Via header field.

Description

Contains the version of the protocol used to send the message. For example, in the header:

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7

this is the token containing "2.0".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.104 - GetViaSentBy

3.4.1.16.3.104.1 - CSipHeader::GetViaSentBy Method

Provides access to the Via header's sent-by construct.

C++

```
const CHostPort& GetViaSentBy() const;
CHostPort& GetViaSentBy();
```

Returns

The host name and port (CHostPort (see page 167)) in the Via header field.

Description

Contains the client's host name or network address, and possibly the port number at which it wants to receive responses. For example, in the header:

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7

this is the host name and port containing "erlang.bell-telephone.com:5060".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.105 - GetViaTransport

3.4.1.16.3.105.1 - CSipHeader::GetViaTransport Method

Provides access to the Via header's transport construct.

C++

```
const CToken& GetViaTransport() const;
CToken& GetViaTransport();
```

Returns

The third token (CToken (see page 360)) in the Via header field.

Description

Contains the name of the transport protocol used to send the message. For example, in the header:

```
Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7
```

this is the token containing "UDP".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.106 - GetWarning

3.4.1.16.3.106.1 - CSipHeader::GetWarning Method

Provides access to the Warning header's warning-value construct.

C++

```
const CToken& GetWarning() const;
CToken& GetWarning();
```

Returns

The token (CToken (see page 360)) in the Warning header field.

Description

Carries additional information about the status of a response. For example, in the header:

```
Warning:307 isi.edu "Session parameter 'foo' not understood"
```

this is the token containing '307 isi.edu "Session parameter 'foo' not understood" '.

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.107 - GetWwwAuthenticateScheme

3.4.1.16.3.107.1 - CSipHeader::GetWwwAuthenticateScheme Method

Provides access to the WWW-Authenticate header's auth-scheme construct.

C++

```
const CToken& GetWwwAuthenticateScheme() const;
CToken& GetWwwAuthenticateScheme();
```

Returns

The token (CToken (see page 360)) in the WWW-Authenticate header field.

Description

Contains an authentication challenge. For example, in the header:

```
WWW-Authenticate: Digest realm="atlanta.com",
  domain="sip:boxesbybob.com", qop="auth",
  nonce="f84f1cec41e6cbe5aea9c8e88d359",
  opaque="", stale=FALSE, algorithm=MD5
```

this is the token containing "Digest".

See Also

Header accessor nomenclature in CSipHeader (see page 239).

3.4.1.16.3.108 - CSipHeader::InsertNextHeader Method

Inserts a header at the given 0-based index in the chain. Index 0 refers to this CSipHeader (see page 239)'s first next header.

C++

```
mxt_result InsertNextHeader(IN unsigned int uIndex, IN TO CSipHeader* pNewHeader, IN bool bTakeOwnershipOnSuccess = false);
```

Parameters

Parameters	Description
IN unsigned int uIndex	0-based index where to add the header. Index 0 refers to this CSipHeader (see page 239)'s first next header. uIndex must be between 0 and GetNbNextHeaders (see page 264)() inclusively. For example, if uIndex is 0, the next index of this CSipHeader (see page 239) becomes pNewHeader. The next header of pNewHeader becomes the header that was the next header of this CSipHeader (see page 239) before this method was called.
IN TO CSipHeader* pNewHeader	Header to add at in the list. The header MUST NOT be chained. If the header is chained, this method returns an error. It MUST NOT be NULL. Ownership is TAKEN.
IN bool bTakeOwnershipOnSuccess = false	<ul style="list-style-type: none"> false: Default value. Ownership of pNewHeader is always taken. true: Ownership is taken if the return value is a success only.

Returns

resS_OK : Header is correctly inserted.
 resFE_INVALID_ARGUMENT : uIndex is an invalid index.
 resFE_INVALID_ARGUMENT : Given header is of a different type than current header.
 resFE_INVALID_ARGUMENT : This type of header cannot have multiple instances within a same SIP packet.
 resFE_INVALID_ARGUMENT : pNewHeader is NULL.
 resFE_INVALID_ARGUMENT : The header contains chained headers.

Description

Inserts a header at the given index of this header type's list. It can be used only for multi or combinable header types (e.g., Via, Contact, etc.). Only headers of the same type can be appended after each other. A chain of raw and parsed headers can be made.

See Also

AppendNextHeader (see page 246), RemoveNextHeader (see page 291), UnlinkNextHeader (see page 295)

3.4.1.16.3.109 - CSipHeader::IsContactWildcard Method

Returns true if the Contact header's value is STAR.

C++

```
bool IsContactWildcard() const;
```

Returns

True if the Contact header is a wildcard, false otherwise.

Description

A Contact header can be a wildcard if it has the value "*". This is used only in REGISTER requests with an Expires header set to "0" to remove all registrations.

3.4.1.16.3.110 - CSipHeader::IsEmptyHeader Method

Returns true if this header is empty, and can be empty as per the header definition table.

C++

```
bool IsEmptyHeader() const;
```

Returns

True if the header is considered empty.

Description

Returns true if this header is empty, and can be empty as per the header definition table.

See Also

IsParsedDataAvailable (see page 290)

3.4.1.16.3.111 - CSipHeader::IsParsedDataAvailable Method

Returns true if parsed data (parsed from raw or set by application) is available.

C++

```
bool IsParsedDataAvailable() const;
```

Returns

True if parsed data is available.

Description

Checks if the CSipHeader (see page 239) contains data in the parsed form.

3.4.1.16.3.112 - CSipHeader::IsSingleHdrEquivalent Method

Returns true if the current instance is equivalent to the cited instance, without regard to the next headers.

C++

```
bool IsSingleHdrEquivalent(IN const CSipHeader& rSrc) const;
```

Parameters

Parameters	Description
IN const CSipHeader& rSrc	Header against which to compare.

Returns

True if both headers are equivalent.

Description

Verifies if the current header is equivalent to the specified source. This method does not check the next headers. To be equivalent, headers must be of the same type and have equivalent parameter lists and contents. An unparsed header can only be equal to itself, never to another parsed or unparsed header.

Warning

This method assumes the headers have been parsed prior to calling it.

See Also

operator==

3.4.1.16.3.113 - CSipHeader::Parse Method

Parses the header chain for up to uMaxHeaders.

C++

```
mxt_result Parse(IN unsigned int uMaxHeaders = uHEADERS_PARSE_ALL);
```

Parameters

Parameters	Description
IN unsigned int uMaxHeaders = uHEADERS_PARSE_ALL	Maximum number of headers that must be parsed during this call. Set to uHEADERS_PARSE_ALL (see page 401) to ask to parse all available headers in the chain.

Returns

resFE_UNEXPECTED : There is no raw data to parse.
 resFE_UNEXPECTED : The parser encountered unexpected data during processing. The current parsed data has been reset.
 resS_OK : Success, reached end of headers to parse.
 resSI_SIPPARSER_DATACONT : Success, there are more headers to parse.
 resSW_SIPPARSER_HEADER_NOT_ENOUGH_HEADERS : Success, but there are not as many headers available as requested.
 This method can also return any error mxt_result that ParseSingleHeader() can return.

Description

This method is used to parse a portion, or all, of the headers in the chain. Initially, the header chain can be made up of raw headers (unparsed data), and/or parsed headers, in any order.

This method first looks if the header is parsed, and if not, it parses it. In the case of raw headers, some may be comma-separated headers, in which case new CSipHeaders are inserted in the header chain at the correct position as parsing of raw headers progresses.

The forementioned process goes on until uMaxHeaders are in the parsed state, or an error has been encountered during the parsing of a raw header. Note that the number of parsed headers (GetNbParsedHeaders (see page 264)()) after the call to this method is equal or greater than uMaxHeaders. For example, if 3 headers are already parsed and Parse(2) is called, there is no action taken except to return a success.

If an error occurs during the parsing of a raw header, the appropriate error code is returned. Previously parsed headers, whether having been parsed in an earlier call or in the same call, remain parsed, as they were already validated. In case of parsing error, the user should consider that the return code for parsing the previous headers was resSI_SIPPARSER_DATACONT. Use the GetNbParsedHeaders (see page 264) and GetNbNextHeaders (see page 264) methods to check if all headers have been correctly parsed (in addition to this method's return code).

Parsing an empty header (no raw data, no parsed data) returns a success result, but only if the header type allows being empty.

3.4.1.16.3.114 - CSipHeader::RemoveNextHeader Method

Removes a header from the chain, at the given 0-based index. Index 0 refers to this CSipHeader (see page 239)'s first next header.

C++

```
mxt_result RemoveNextHeader(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	0-based index of the header to remove. Index 0 refers to this CSipHeader (see page 239)'s first next header.

Returns

resS_OK : Header removed.
 resFE_INVALID_ARGUMENT : Index is out of range.

Description

Removes a header from the chain of headers. Note that this method acts on this instance's list of next header, thus excluding the current instance.

See Also

AppendNextHeader (see page 246)

3.4.1.16.3.115 - CSipHeader::ReplaceNextHeader Method

Replaces the next headers with the headers in parameter.

C++

```
mxt_result ReplaceNextHeader(IN TO CSipHeader* pNewHeader);
```

Parameters

Parameters	Description
IN TO CSipHeader* pNewHeader	Header to replace the list of next headers. Ownership is TAKEN. It can be NULL.

Returns

resS_OK : Header is correctly set.

resFE_INVALID_ARGUMENT : Given header is of a different type than current header.

resFE_INVALID_ARGUMENT : This type of header cannot have multiple instances within a same SIP packet.

Description

Replaces the next header list with the header in parameter.

3.4.1.16.3.116 - CSipHeader::Reset Method

Clears all parsed/Set data.

C++

```
void Reset(IN EResetLevel = eCOMPLETE_RESET);
```

3.4.1.16.3.117 - CSipHeader::Serialize Method

Outputs the body of the header into the blob.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Container for the output data.

Description

Outputs the content of the header into the CBlob. This method only outputs the header body, not the name, but does output the optional parameter list. If raw data is available, it is output instead of outputting the parsed data because this preserves the incoming format.

3.4.1.16.3.118 - CSipHeader::SetContactWildcard Method

Sets the value of the STAR for the Contact header's (name-addr / addr-spec) construct.

C++

```
void SetContactWildcard(IN bool bIsWildcard);
```

Parameters

Parameters	Description
IN bool bIsWildcard	Set to true to indicate the wildcard.

Description

Uses the parsed data structure as if it was of the SNameAddrForm form. Sets the wildcard boolean. Note that the name-addr member of the SNameAddrForm form and the wildcard are mutually exclusive.

Warning

The user of the CSipHeader (see page 239) is required to verify the header type before calling this helper. In the interest of keeping the code size small, no verifications are made versus m_eHeader or the current real use of the data structure union in these methods. It is thus VERY IMPORTANT that CSipHeader (see page 239) users make sure that they are using the right type of interface!

See Also

IsContactWildcard (see page 289)

3.4.1.16.3.119 - CSipHeader::SetNameForm Method

Sets the form of header name to use for all header types.

C++

```
static void SetNameForm( IN EHeaderNameForm eHeaderNameForm );
```

Parameters

Parameters	Description
bEnableShortForm	Setting for the short name form. If true, short names are preferred when GetName() is called.

Description

Tells the M5T SIP stack parser to prefer short names when serializing headers for output to the network.

See Also

GetName, HeaderDefinition

3.4.1.16.3.120 - SetParam**3.4.1.16.3.120.1 - CSipHeader::SetParam Method**

Sets a header parameter value to the parameter list from a CString.

C++

```
mxt_result SetParam( IN const char* szName, IN const CString& rstrValue );
```

Parameters

Parameters	Description
IN const char* szName	Name of the parameter to set. A pre-defined list of parameters exists in SipParser/HeaderParameter.h.
IN const CString& rstrValue	Value of the parameter to set.

Returns

resS_OK : The parameter is properly set in the parameter list.

resFE_UNEXPECTED : This header cannot have parameters.

Description

Sets this header's parameter list with the szName parameter. The parameter is set with the value held by rstrValue.

This method takes care of the following:

- If this header has no parameter list yet, one is created.
- If a szName parameter already exists, its value is replaced with the value held by rstrValue.
- If no szName parameter already exists, it is created and appended to the parameter list.

To set a parameter with no value, this method must be called with an empty CString for rstrValue.

See Also

[GetParamList](#) (see page 267), [SetParamList](#) (see page 294)

3.4.1.16.3.120.2 - CSipHeader::SetParam Method

Sets a header parameter value to the parameter list from a CToken (see page 360).

C++

```
mxt_result SetParam(IN const char* szName, IN const CToken& rValue);
```

Parameters

Parameters	Description
IN const char* szName	Name of the parameter to set. A pre-defined list of parameters exists in SipParser/HeaderParameter.h.
IN const CToken& rValue	Value of the parameter to set.

Returns

resS_OK : The parameter is properly set in the parameter list.

resFE_UNEXPECTED : This header cannot have parameters.

Description

Sets this header parameter list with the `szName` parameter. The parameter is set with the value held by `rValue`.

This method takes care of the following:

- If this header has no parameter list yet, one is created.
- If a `szName` parameter already exists, its value is replaced with the value held by `rValue`.
- If no `szName` parameter already exists, it is created and appended to the parameter list.

See Also

[GetParamList](#) (see page 267), [SetParamList](#) (see page 294)

3.4.1.16.3.121 - CSipHeader::SetParamList Method

Sets the parameter list.

C++

```
mxt_result SetParamList(IN TO CGenParamList* pList);
```

Parameters

Parameters	Description
IN TO CGenParamList* pList	List to set. Ownership is TAKEN.

Returns

resS_OK : Pointer set.

resFE_UNEXPECTED : This header cannot have parameters as per the header definition table.

Description

Sets the parameter list. If a list is already present, it is replaced by the new one.

See Also

[GetParamList](#) (see page 267)

3.4.1.16.3.122 - CSipHeader::SetRawHeader Method

Sets the raw header. Ownership is taken.

C++

```
mxt_result SetRawHeader(IN TO CRawHeader* pRawHeader);
```

Parameters

Parameters	Description
IN TO CRawHeader* pRawHeader	Raw header of which to take ownership. Ownership is TAKEN.

Returns

resS_OK : Header set.

resFE_INVALID_ARGUMENT : The extension header name must be set before setting the raw header. Use GetExtensionHeaderName (see page 258)().

Description

This method overrides the current raw header with the new one provided. If the header current type is eHDR_EXTENSION, the raw header must have a valid non-empty name.

To reset the raw header only, call SetRawHeader(NULL).

Warning

The extension header name must be set before setting the raw header by using the GetExtensionHeaderName (see page 258)() method.

See Also

GetRawHeader (see page 274)

3.4.1.16.3.123 - CSipHeader::UnlinkNextHeader Method

Removes a header from the chain and returns it, at the given 0-based index. Index 0 refers to this CSipHeader (see page 239)'s first next header.

C++

```
GO CSipHeader* UnlinkNextHeader(IN unsigned int uIndex);
```

Parameters

Parameters	Description
IN unsigned int uIndex	0-based index of the header to remove. Index 0 refers to this CSipHeader (see page 239)'s first next header. For example, if uIndex is 0, the unlinked header's next header is changed to NULL and the header it previously had as next header is set as this header's next header.

Returns

The pointer to the unlinked header. The ownership is GIVEN to the caller. The unlinked header does not have a next header.

NULL if uIndex is out of range.

Description

Removes a header from the chain of headers and returns it to the caller. Note that this method acts on this instance's list of next header, thus excluding the current instance.

The header returned to the called has no next header. The header that was its next header is chained at its position.

See Also

RemoveNextHeader (see page 291), AppendNextHeader (see page 246)

3.4.1.16.3.124 - CSipHeader::UnlinkTopHeader Method

Unlinks the top header from the chain. The returned header is the chain excluding the current header.

C++

```
GO CSipHeader* UnlinkTopHeader();
```

Returns

A CSipHeader (see page 239)*, the value previously stored in m_pNextHeader.

The ownership of the object is GIVEN to the caller.

Description

Sets the m_pNextHeader pointer to NULL. The m_pNextHeader is not deleted so the caller must be carefull not to lose the return value.

See Also

RemoveNextHeader (see page 291), GetNextHeader (see page 265)

3.4.1.16.4 - Operators**3.4.1.16.4.1 - CSipHeader::!= Operator**

Comparison operator.

C++

```
bool operator !=(INOUT CSipHeader& rSrc);
```

Parameters

Parameters	Description
INOUT CSipHeader& rSrc	Source against which to compare.

Returns

True if the header chains are different.

Description

Inequality operator.

See Also

operator==

3.4.1.16.4.2 - CSipHeader::= Operator

Assignment operator.

C++

```
CSipHeader& operator =(IN const CSipHeader& rSrc);
```

Parameters

Parameters	Description
IN const CSipHeader& rSrc	Source from which to copy.

Returns

Returns a reference on "this" instance.

Description

Assignment operator.

3.4.1.16.4.3 - CSipHeader::== Operator

Comparison operator.

C++

```
bool operator ==(INOUT CSipHeader& rSrc);
```

Parameters

Parameters	Description
INOUT CSipHeader& rSrc	Source against which to compare.

Returns

True if both header chains are equivalent.

Description

Comparison operator. Header chains are equivalent if they have the same number of headers, and the headers that they contain are all equivalent. The order of these headers is not important.

See Also

IsSingleHeaderEquivalent, CompareHeaderChain

3.4.1.16.5 - Enumerations**3.4.1.16.5.1 - CSipHeader::EHeaderNameForm Enumeration**

```
enum EHeaderNameForm {
    eSHORT_FORM_NAME,
    eLONG_FORM_NAME
};
```

Description

Indicates to use the short or long header form.

Members

Members	Description
eSHORT_FORM_NAME	Use the short header form.
eLONG_FORM_NAME	Use the long header form.

3.4.1.16.5.2 - CSipHeader::EParameterCreationBehavior Enumeration

```
enum EParameterCreationBehavior {
    ePARAM_DONT_CREATE,
    ePARAM_CREATE_NEW
};
```

Description

Indicates whether or not the parameter must be created when calling GetParam (see page 266).

Members

Members	Description
ePARAM_DONT_CREATE	Dont' create the parameter.
ePARAM_CREATE_NEW	Create the parameter.

3.4.1.16.5.3 - CSipHeader::EResetLevel Enumeration

```
enum EResetLevel {
    eBASIC_RESET = 0x1000,
    eNEXT_HDRS_RESET = ((int)eBASIC_RESET | 0x1),
    eRAW_HDR_RESET = ((int)eBASIC_RESET | 0x2),
    eCOMPLETE_RESET = ((int)eBASIC_RESET | eNEXT_HDRS_RESET | eRAW_HDR_RESET)
};
```

Description

Indicate what to reset to the Reset (see page 292) method.

Members

Members	Description
eBASIC_RESET = 0x1000	Resets the parameter list and the header values.
eNEXT_HDRS_RESET = ((int)eBASIC_RESET 0x1)	Does basic reset in addition of resetting the m_pNextHeader member.
eRAW_HDR_RESET = ((int)eBASIC_RESET 0x2)	Does basic reset in addition of resetting the m_pRawHeader member.
eCOMPLETE_RESET = ((int)eBASIC_RESET eNEXT_HDRS_RESET eRAW_HDR_RESET)	Resets all

3.4.1.17 - CSipMessageBody Class

Class Hierarchy

```
CSipMessageBody
```

C++

```
class CSipMessageBody;
```

Description

The CSipMessageBody is used to represent one or more bodies that may be included by a SIP packet. This class is used when sending and receiving SIP packets.

One or more body can be configured in an instance of this class. When a single body is configured, this class represents this body as it was configured. For example, a body of type "application/sdp" is represented as "application/sdp". When multiple bodies are configured in an instance of this class, they are then represented as MIME objects.

In order to parse or serialize a MIME object, the application MUST use the CSipPacketParser (see page 312) class. These methods are respectively CreateSipMessageBody and Serialize (see page 310).

Creating Bodies

There are two ways to add a body to an instance of this class. The first is AddBody (see page 306), which takes a CBlob as a parameter. This method adds a body to the level that is being represented by this instance. The second AddBody (see page 306) method takes a CSipMessageBody as a parameter. This additional body is inserted as an "inner" body to the current body. Refer to the example section to understand this concept of "inner" body and insertion at the current level.

When configuring bodies, this class automatically manages the Content-Length header according to the size of the configured bodies. The user of this class never has to configure a Content-Length header.

Accessing Existing Bodies

To access the bodies, you first need to parse them. This is done by calling CSipPacket::CreateSipMessageBody, which calls ParseBody, giving it the packet to parse. Once this is done, GetNumberOfBodies (see page 309) gives the number of bodies at the current level. GetExternalMimeHeaders (see page 308) returns the headers that are external to the current level. Once the number of bodies is known, the application can get the MIME headers and content associated with each of them. The content is obtained via GetBlobBody (see page 307) and the headers via GetMimeHeaders (see page 309). The application is responsible to parse the bodies obtained via the GetBlobBody (see page 307) method. It can use whatever means to parse the body. If GetBlobBody (see page 307) returns NULL for a specific index, it means that there is an inner message-body. The message-body is obtained via GetSipMessageBody (see page 309). In case there no inner body for a specific index, GetSipMessageBody (see page 309) returns NULL and GetBlobBody (see page 307) non-NULL. Refer to the example section to understand this concept of "inner" body and fetching from the current level.

MIME is defined in RFCs 2045, 2046, and 2047.

Location

SipParser/CSipMessageBody.h

Example

Assume you need to create the following body for an INVITE to be sent:

```
// Sample INVITE taken from RFC 3893.
//-----

INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/UDP pc33.example.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@example.net>
From: Alice <sip:alice@example.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.example.com>
Content-Type: multipart/mixed; boundary=unique-boundary-1

Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 example.com
s=Session SDP
c=IN IP4 pc33.example.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42
Content-Length: 608

Content-Type: message/sipfrag
Content-Disposition: aib; handling=optional

From: Alice <sip:alice@example.com>
To: Bob <sip:bob@example.net>
Contact: <sip:alice@pc33.example.com>
Date: Thu, 21 Feb 2002 13:02:03 GMT
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE

Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
  handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jh77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756
```

The following pseudo-code would need to be implemented:

```
1- Create 3 CSipMessageBody instances:
  -> bodyInner2, for the SIP Authenticated Body (sipfrag and signature)
  -> bodyInner, for the SDP and the signed AIB
  -> bodyOuter, that will contain all bodies.

2- Start by configuring bodyInner with its two bodies:
  -> bodySipFrag, Create a new message body for the sipfrag.
  -> Create CSipHeader for Content-Type (pHdrContentTypeSipFrag)
  -> Configure pHdrContentTypeSipFrag with "message/sipfrag"
  -> bodySipFrag.AddBody(blobSipFrag, TO pHdrContentTypeSipFrag);

  -> bodySignature, Create a new message body for the signature.
  -> Create CSipHeader for Content-Type (pHdrContentTypeSignature)
  -> Configure pHdrContentTypeSignature with "application/pkcs7-signature"
  -> Configure additional parameters in pHdrContentTypeSignature.
  -> Create CHeaderList that will contain extra header to add as
```

```

extra header (pExtraHeader).
-> bodySignature, Create a new message body for the signature.
-> bodySignature.AddBody(bodySignature,
    TO pHdrContentTypeSignature,
    TO pExtraHeader);

-> bodyInner2.AddBody(bodySipFrag);
-> bodyInner2.AddBody(bodySignature);

3- Set the Content-Type of bodyInner2:
-> Create CSipHeader for Content-Type (pHdrContentTypeInner)
-> Configure pHdrContentTypeInner with "multipart/signed"
-> Configure additional parameters in pHdrContentTypeInner
-> bodyInner2.SetExternalMimeHeaders(TO pHdrContentTypeInner);

4- Configure bodyInner with its two bodies:
-> bodySdp, Create a new message body for the SDP.
-> Create CSipHeader for Content-Type (pHdrContentTypeSdp)
-> Configure pHdrContentTypeSdp with "application/sdp"
-> bodySdp.AddBody(blobSDP, TO pHdrContentTypeSdp);

-> bodyInner.AddBody(bodySdp);
-> bodyInner.AddBody(bodyInner2);

5- Add the bodyInner and bodyInner2 to the bodyOuter.
-> bodyOuter.AddBody(bodyInner);
-> bodyOuter.AddBody(bodyInner2);

6- Set the Content-Type of bodyOuter:
-> Create CSipHeader for Content-Type (pHdrContentTypeOuter)
-> Configure pHdrContentTypeOuter with "multipart/mixed"
-> bodyOuter.SetExternalMimeHeaders(TO pHdrContentTypeOuter);

7- Pass bodyOuter as a parameter to the ISipSessionSvc::Invite() to send
the proper payload.

```

Assume you need to parse the same body for an INVITE that has been received. The following pseudo-code would need to be implemented:

```

1- Parse the message-body:
-> body <- rPacket.CreateSipMessageBody();
-> body.GetExternalMimeHeaders();

2- Get the number of bodies:
-> body.GetNumberOfBodies() which returns 2

3- Get the body and MIME headers for first index starting at zero
-> body.GetBlobBody(0)
-> body.GetMimeHeaders(0)

4- Get the body for second index:
-> body.GetBlobBody(1) but it returns NULL meaning that it is an inner
body at the current level.
-> body.GetSipMessageBody(1) which gives innerBody

5- Get the external MIME headers for this inner body:
-> innerBody.GetExternalMimeHeaders();

6- Get the innerBody bodies and headers:
-> innerBody.GetNumberOfBodies() which also returns 2
-> innerBody.GetMimeHeaders(0);
-> innerBody.GetBlobBody(0)
-> innerBody.GetMimeHeaders(1);
-> innerBody.GetBlobBody(1)

```

RFC 2045 ABNF:

```

attribute := token
    ; Matching of attributes
    ; is ALWAYS case-insensitive.

composite-type := "message" / "multipart" / extension-token

content := "Content-Type" ":" type "/" subtype
    *( ";" parameter)
    ; Matching of media type and subtype
    ; is ALWAYS case-insensitive.

```

```

description := "Content-Description" ":" *text

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

encoding := "Content-Transfer-Encoding" ":" mechanism

entity-headers := [ content CRLF ]
                  [ encoding CRLF ]
                  [ id CRLF ]
                  [ description CRLF ]
                  *( MIME-extension-field CRLF )

extension-token := ietf-token / x-token

hex-octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
; Octet must be used for characters > 127, =
; SPACES or TABs at the ends of lines, and is
; recommended for any character not listed in
; RFC 2049 as "mail-safe".

iana-token := <A publicly-defined extension token. Tokens
              of this form must be registered with IANA
              as specified in RFC 2048.>

ietf-token := <An extension token defined by a
              standards-track RFC and registered
              with IANA.>

id := "Content-ID" ":" msg-id

mechanism := "7bit" / "8bit" / "binary" /
                 "quoted-printable" / "base64" /
                 ietf-token / x-token

MIME-extension-field := <Any RFC 822 header field which
                           begins with the string
                           "Content-"

MIME-message-headers := entity-headers
                        fields
                        version CRLF
                        ; The ordering of the header
                        ; fields implied by this BNF
                        ; definition should be ignored.

MIME-part-headers := entity-headers
                     [fields]
                     ; Any field not beginning with
                     ; "content-" can have no defined
                     ; meaning and may be ignored.
                     ; The ordering of the header
                     ; fields implied by this BNF
                     ; definition should be ignored.

parameter := attribute "=" value

ptext := hex-octet / safe-char

qp-line := *(qp-section transport-padding CRLF)
           qp-part transport-padding

qp-part := qp-section
           ; Maximum length of 76 characters

qp-section := [* (ptext / SPACE / TAB) ptext]

qp-segment := qp-section *(SPACE / TAB) "="
           ; Maximum length of 76 characters

quoted-printable := qp-line *(CRLF qp-line)

safe-char := <any octet with decimal value of 33 through
            60 inclusive, and 62 through 126>
            ; Characters not listed as "mail-safe" in
            ; RFC 2049 are also not recommended.

```

```

subtype := extension-token / iana-token

token := 1*char
                    ; Composers MUST NOT generate
                    ; non-zero length transport
                    ; padding, but receivers MUST
                    ; be able to handle padding
                    ; added by message transports.

tspecials := "(" / ")" / "<" / ">" / "@" /
             "," / ";" / ":" / "" / "<">
             "/" / "[" / "]" / "?" / "="
             ; Must be in quoted-string,
             ; to use within parameter values

type := discrete-type / composite-type

value := token / quoted-string
version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT

x-token := <The two characters "X-" or "x-" followed, with
          no intervening white space, by any token>

RFC 2392 ABNF:

msg-id      = url-addr-spec

url-addr-spec = addr-spec ; URL encoding of RFC 822 addr-spec

RFC 2046 ABNF:

boundary := 0*69<bchars> bcharsnospace

bchars := bcharsnospace / " "

bcharsnospace := DIGIT / ALPHA / "!" / "(" / ")" /
                  "+" / "_" / "," / "-" / "." /
                  "/" / ":" / "=" / "?"

dash-boundary := "--" boundary
                 ; boundary taken from the value of
                 ; boundary parameter of the
                 ; Content-Type field.

multipart-body := [preamble CRLF]
                  dash-boundary transport-padding CRLF
                  body-part *encapsulation
                  close-delimiter transport-padding
                  [CRLF epilogue]

transport-padding := *LWSP-char
                    ; Composers MUST NOT generate
                    ; non-zero length transport
                    ; padding, but receivers MUST
                    ; be able to handle padding
                    ; added by message transports.

encapsulation := delimiter transport-padding
                  CRLF body-part

delimiter := CRLF dash-boundary

close-delimiter := delimiter "--"

preamble := discard-text

epilogue := discard-text

discard-text := *(*text CRLF) *text
                 ; May be ignored or discarded.

body-part := MIME-part-headers [CRLF *OCTET]

```

```

; Lines in a body-part must not start
; with the specified dash-boundary and
; the delimiter must not appear anywhere
; in the body part. Note that the
; semantics of a body-part differ from
; the semantics of a message, as
; described in the text.

OCTET := <any 0-255 octet value>

RFC 1847 ABNF: this RFC defines the Content-Type headers for multipart
signed and encrypted contents

Definition of Multipart/Signed

(1) MIME type name: multipart
(2) MIME subtype name: signed
(3) Required parameters: boundary, protocol, and micalg

boundary := "boundary" "=" value
parameter := "protocol" "=" value
value := <"> type "/" subtype <">
parameter := "micalg" "=" value

Definition of Multipart/Encrypted

(1) MIME type name: multipart
(2) MIME subtype name: encrypted
(3) Required parameters: boundary, protocol

boundary := "boundary" "=" value
parameter := "protocol" "=" value
value := <"> type "/" subtype <">

```

Constructors

Constructor	Description
CSipMessageBody (see page 306)	Default Constructor.

Legend

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

Destructors

Destructor	Description
~CSipMessageBody (see page 306)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
= (see page 311)	Assignment operator.

Legend

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

Methods

Method	Description
AddBody (see page 306)	Appends a body at the level represented by this instance.
GetBlobBody (see page 307)	Gets the body for this level.
GetExternalMimeHeaders (see page 308)	Provides access to the mime headers for this level.
GetMimeHeaders (see page 309)	Gets the mime headers associated with the specified body.
GetNumberOfBodies (see page 309)	Gets the number of bodies held by this level.
GetSipMessageBody (see page 309)	Gets the body at unIndex, assuming it is a recursive CSipMessageBody.
Reset (see page 310)	Clears all data back to its initial state.

• Serialize (see page 310)	If the Content-Type of this body is multipart, sets the outer level boundary and serializes this body information into a blob.
• SetExternalMimeHeaders (see page 310)	Configures the information pertaining to the body at this level.

Legend

•	Method
---	--------

3.4.1.17.1 - Data Members

3.4.1.17.1.1 - CSipMessageBody::ms_pszBOUNDARY_OUTER_LEVEL Data Member

The default value for most outer body boundary, this is "level-0".

C++

```
const char* const ms_pszBOUNDARY_OUTER_LEVEL;
```

3.4.1.17.1.2 - CSipMessageBody::ms_pszCONTENT_TYPE_APPLICATION_MEDIA_TYPE Data Member

The "application" Content-Type media-type value.

C++

```
const char* const ms_pszCONTENT_TYPE_APPLICATION_MEDIA_TYPE;
```

3.4.1.17.1.3 - CSipMessageBody::ms_pszCONTENT_TYPE_DIGEST_MEDIA_SUBTYPE Data Member

The "digest" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_DIGEST_MEDIA_SUBTYPE;
```

3.4.1.17.1.4 - CSipMessageBody::ms_pszCONTENT_TYPE_MESSAGE_MEDIA_TYPE Data Member

The "message" Content-Type media-type value.

C++

```
const char* const ms_pszCONTENT_TYPE_MESSAGE_MEDIA_TYPE;
```

3.4.1.17.1.5 - CSipMessageBody::ms_pszCONTENT_TYPE_MIXED_MEDIA_SUBTYPE Data Member

The "mixed" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_MIXED_MEDIA_SUBTYPE;
```

3.4.1.17.1.6 - CSipMessageBody::ms_pszCONTENT_TYPE_MULTIPART_MEDIA_TYPE Data Member

The "multipart" Content-Type media-type value.

C++

```
const char* const ms_pszCONTENT_TYPE_MULTIPART_MEDIA_TYPE;
```

3.4.1.17.1.7 -**CSipMessageBody::ms_pszCONTENT_TYPE_PKCS7_SIGNATURE_MEDIA_SUBTYPE Data Member**

The "pkcs7-signature" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_PKCS7_SIGNATURE_MEDIA_SUBTYPE;
```

3.4.1.17.1.8 - CSipMessageBody::ms_pszCONTENT_TYPE_PLAIN_MEDIA_SUBTYPE Data Member

The "plain" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_PLAIN_MEDIA_SUBTYPE;
```

3.4.1.17.1.9 - CSipMessageBody::ms_pszCONTENT_TYPE_RFC822_MEDIA_SUBTYPE Data Member

The "rfc822" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_RFC822_MEDIA_SUBTYPE;
```

3.4.1.17.1.10 - CSipMessageBody::ms_pszCONTENT_TYPE_SDP_MEDIA_SUBTYPE Data Member

The "sdp" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_SDP_MEDIA_SUBTYPE;
```

3.4.1.17.1.11 - CSipMessageBody::ms_pszCONTENT_TYPE_SIGNED_MEDIA_SUBTYPE Data Member

The "signed" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_SIGNED_MEDIA_SUBTYPE;
```

3.4.1.17.1.12 - CSipMessageBody::ms_pszCONTENT_TYPE_SIP_MEDIA_SUBTYPE Data Member

The "sip" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_SIP_MEDIA_SUBTYPE;
```

3.4.1.17.1.13 - CSipMessageBody::ms_pszCONTENT_TYPE_SIPFRAG_MEDIA_SUBTYPE Data Member

The "sipfrag" Content-Type media-subtype value.

C++

```
const char* const ms_pszCONTENT_TYPE_SIPFRAG_MEDIA_SUBTYPE;
```

3.4.1.17.1.14 - CSipMessageBody::ms_pszCONTENT_TYPE_TEXT_MEDIA_TYPE Data Member

The "text" Content-Type media-type value.

C++

```
const char* const ms_pszCONTENT_TYPE_TEXT_MEDIA_TYPE;
```

3.4.1.17.2 - Constructors

3.4.1.17.2.1 - CSipMessageBody

3.4.1.17.2.1.1 - CSipMessageBody::CSipMessageBody Constructor

Default Constructor.

C++

```
CSipMessageBody();
```

Description

Default constructor.

3.4.1.17.2.1.2 - CSipMessageBody::CSipMessageBody Constructor

Copy Constructor.

C++

```
CSipMessageBody( IN const CSipMessageBody& rSrc );
```

Parameters

Parameters	Description
IN const CSipMessageBody& rSrc	Message-body from which to initialize.

Description

Copy constructor.

3.4.1.17.3 - Destructors

3.4.1.17.3.1 - CSipMessageBody::~CSipMessageBody Destructor

Destructor.

C++

```
virtual ~CSipMessageBody();
```

Description

Default destructor.

3.4.1.17.4 - Methods

3.4.1.17.4.1 - AddBody

3.4.1.17.4.1.1 - CSipMessageBody::AddBody Method

Appends a body at the level represented by this instance.

C++

```
mxt_result AddBody(IN TO CBlob* pBlobBody, IN TO CSipHeader* pHeaderContentType, IN TO CHeaderList* pAdditionalMimeHeaders = NULL);
```

Parameters

Parameters	Description
IN TO CBlob* pBlobBody	The blob to set. Ownership is TAKEN.
IN TO CSipHeader* pHeaderContentType	The Content-Type header describing the content to add. Ownership is TAKEN.
IN TO CHeaderList* pAdditionalMimeHeaders = NULL	Additional headers added with the body within its boundaries. Ownership is TAKEN.

Returns

resS_OK: The body is added.

resFE_FAIL: pHeaderContentType is not a Content-Type header or the blob is NULL or it has a size of zero.

Description

Sets a blob body to the current level of bodies. This will be the outer body. Note that you must not set a blob to the current level if the message body contains inner bodies.

No verification on the headers provided in pAdditionalMimeHeaders is done, however only headers of the following types should be specified:

- MIME-Version
- Content-Description
- Content-Disposition
- Content-ID
- Content-Encoding
- Content-Language
- Content-Transfer-Encoding

3.4.1.17.4.1.2 - CSipMessageBody::AddBody Method

Appends an inner body at the level represented by this instance.

C++

```
mxt_result AddBody(IN TO CSipMessageBody* pInnerBody);
```

Parameters

Parameters	Description
IN TO CSipMessageBody* pInnerBody	The inner body to add. Ownership is TAKEN.

Returns

resS_OK: The inner body is added.

resFE_FAIL: pInnerBody is NULL

Description

Adds an inner body to the level of bodies. This inner body has its own list of bodies at its level.

3.4.1.17.4.2 - GetBlobBody**3.4.1.17.4.2.1 - CSipMessageBody::GetBlobBody Method**

Gets the body for this level.

C++

```
CBlob* GetBlobBody();
```

Returns

The blob body or NULL if there is no content.

Description

Gets the body for this level. The application is responsible to parse this header.

3.4.1.17.4.2.2 - CSipMessageBody::GetBlobBody Method

Gets the body for this level.

C++

```
const CBlob* GetBlobBody() const;
```

Returns

The blob body or NULL if there is no content

Description

Gets the body for this level. The application is responsible to parse this header.

3.4.1.17.4.2.3 - CSipMessageBody::GetBlobBody Method

Gets the body at uIndex, assuming it is a blob.

C++

```
const CBlob* GetBlobBody(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the message-body, starting at zero.

Returns

The blob content of this body. It could be NULL if this message-body has inner bodies. In that case, GetNumberOfBodies (see page 309) returns higher than zero. It is also NULL if the specified index does not exist.

Description

Gets the blob content of this message-body. The application is responsible to parse its content.

See Also

GetNumberOfBodies (see page 309), GetSipMessageBody (see page 309)

3.4.1.17.4.3 - CSipMessageBody::GetExternalMimeHeaders Method

Provides access to the mime headers for this level.

C++

```
const CHeaderList* GetExternalMimeHeaders() const;
```

Returns

The headers list associated with the current level of this message-body. It could be NULL if no headers are set or none are found when parsing.

Description

Gets the headers that are external to this message-body. It could be NULL if no headers are parsed or set.

See Also

[SetExternalMimeHeaders](#) (see page 310)

3.4.1.17.4.4 - CSipMessageBody::GetMimeHeaders Method

Gets the mime headers associated with the specified body.

C++

```
const CHeaderList* GetMimeHeaders( IN unsigned int uIndex ) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the message-body, starting at zero.

Returns

The headers list associated with a specified index. It could be NULL if there are no MIME headers that were parsed or set or if the specified index does not exist.

Description

Gets the MIME headers list attached to the specified message-body.

Usually headers are of the following types:

- Content-Type
- MIME-Version
- Content-Description
- Content-Disposition
- Content-ID
- Content-Encoding
- Content-Language
- Content-Transfer-Encoding

3.4.1.17.4.5 - CSipMessageBody::GetNumberOfBodies Method

Gets the number of bodies held by this level.

C++

```
unsigned int GetNumberOfBodies() const;
```

Returns

The number of bodies attached to another nested level. It could be zero if no bodies are nested to this message-body.

Description

Gets the number of nested bodies in this message-body. It could be zero, in that case GetBlobBody (see page 307) it returns non-NULL.

See Also

[GetBlobBody](#) (see page 307)

3.4.1.17.4.6 - CSipMessageBody::GetSipMessageBody Method

Gets the body at unIndex, assuming it is a recursive CSipMessageBody (see page 298).

C++

```
const CSipMessageBody* GetSipMessageBody(IN unsigned int uIndex) const;
```

Parameters

Parameters	Description
IN unsigned int uIndex	The index of the message-body, starting at zero.

Returns

The message-body associated with a specified index. It could be NULL if the specified index does not exist.

Description

Gets the message-body for the specified index. All members associated with the returned message-body can be obtained through this class accessors.

3.4.1.17.4.7 - CSipMessageBody::Reset Method

Clears all data back to its initial state.

C++

```
void Reset();
```

Description

Resets the object back to its initial state.

3.4.1.17.4.8 - CSipMessageBody::Serialize Method

If the Content-Type of this body is multipart, sets the outer level boundary and serializes this body information into a blob.

C++

```
void Serialize(INOUT CBlob& rBlob);
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	The blob to serialize.

Description

If the Content-Type of this body is multipart, sets the outer level boundary and serializes this body information into a blob. It outputs the MIME headers, the boundary and the blob content.

If the Content-Type is not multipart, this method does nothing.

See Also

CSipPacketParser::SerializeHelper

3.4.1.17.4.9 - CSipMessageBody::SetExternalMimeHeaders Method

Configures the information pertaining to the body at this level.

C++

```
mxt_result SetExternalMimeHeaders(IN TO CSipHeader* pHeaderContentType, IN TO CHeaderList* pAdditionalMimeHeaders = NULL);
```

Parameters

Parameters	Description
IN TO CSipHeader* pHeaderContentType	The Content-Type header describing the content type to add at the current level. Ownership is TAKEN in case of success.

IN TO CHeaderList* pAdditionalMimeHeaders = NULL	Additional MIME headers that are external to the current level. Ownership is TAKEN in case of success. It can be NULL.
--	--

Returns

resS_OK: Added the headers.

resFE_FAIL: pHeaderContentType is not a Content-Type header.

Description

Sets the headers that are external to this message-body current level. These headers are found right before the first boundary of this message-body in the SIP packet.

Usually, headers found in pAdditionalMimeHeaders are of the following types:

- MIME-Version
- Content-Description
- Content-Disposition
- Content-ID
- Content-Encoding
- Content-Language
- Content-Transfer-Encoding

See Also

GetExternalMimeHeaders (see page 308)

3.4.1.17.5 - Operators**3.4.1.17.5.1 - CSipMessageBody::= Operator**

Assignment operator.

C++

```
CSipMessageBody& operator =(IN const CSipMessageBody& rSrc);
```

Parameters

Parameters	Description
IN const CSipMessageBody& rSrc	Message-body from which to initialize.

Returns

A reference on "this" instance.

Description

Assignment operator.

3.4.1.17.6 - Friends**3.4.1.17.6.1 - friend class CSipPacketParser Friend**

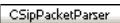
Needed so the CSipPacketParser (see page 312) can access SetBoundary, ParseBody, Serialize (see page 310) and non-const GetExternalMimeHeaders (see page 308) protected methods.

C++

```
friend class CSipPacketParser;
```

3.4.1.18 - CSipPacketParser Class

Class Hierarchy



C++

```
class CSipPacketParser;
```

Description

This class handles entire SIP packets, except for the optional payload. The packet-origination use case typically sets a request or status line, uses the header list to set the data, and then serializes the packet. The packet reception use case repeatedly uses the AppendRawData (see page 314)() method, commits the raw headers, and then works with the header list to parse the received headers.

This class does not parse the optional payload ([message-body]), but carries and serializes it.

```
RFC 3261 ABNF:
generic-message  =  start-line
                  *message-header
                  CRLF
                  [ message-body ]
```

Location

SipParser/CSipPacketParser.h

See Also

CHederList (see page 155), CRawHeader (see page 218), CSipHeader (see page 239), CSipStatusLine (see page 321), CRequestLine (see page 234)

Constructors

Constructor	Description
CSipPacketParser (see page 313)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CSipPacketParser (see page 314)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
= (see page 321)	Assignment operator.

Legend

	Method
--	--------

Methods

Method	Description
AppendRawData (see page 314)	Buffers socket-received data. It also tries to parse the start line if not set. Input must be a NULL-terminated string.
CommitRawDataList (see page 315)	Instructs CSipPacketParser to use its list of buffered raw headers and build its CHederList (see page 155) with them.
CreateSipMessageBody (see page 315)	Creates the message-body, it can be NULL.
GetHeaderList (see page 316)	Provides access to the header list.

◆ GetPayload (See page 316)	Provides access to the optional payload.
◆ GetRawDataList (See page 316)	Provides access to the list of buffered headers.
◆ GetRequestLine (See page 317)	Provides access to the request line if available.
◆ GetSipMessageBody (See page 317)	Gets the message-body. Gets the message-body.
◆ GetStatusLine (See page 318)	Provides access to the status line if available.
◆ IsLocallyGenerated (See page 318)	Returns true if the packet is locally generated.
◆ IsRequest (See page 318)	Returns true if the packet is a request.
◆ IsResponse (See page 319)	Returns true if the packet is a response.
◆ Reset (See page 319)	Clears all data back to its initial state.
◆ Serialize (See page 319)	Outputs the packet's content into the buffer.
◆ SetLocallyGenerated (See page 319)	Sets the local packet generation indicator.
◆ SetPayload (See page 320)	Sets the optional payload.
◆ SetRequestLine (See page 320)	Sets the request line. Converts this packet to a request.
◆ SetSipMessageBody (See page 320)	Sets the message-body.
◆ SetStatusLine (See page 320)	Sets the status line. Converts this packet to a response.

Legend

◆	Method
---	--------

Structs

Struct	Description
SRawData (See page 313)	

3.4.1.18.1 - Structs

3.4.1.18.1.1 - CSipPacketParser::SRawData Struct

```
struct SRawData {
    CRawHeader* m_pRawStartLine;
    CVector<CRawHeader*>* m_pvecpRawData;
};
```

Description

Structure containing the temporary list of raw headers used when collecting received data.

Members

Members	Description
CRawHeader* m_pRawStartLine;	Contains the raw start line.
CVector<CRawHeader*>* m_pvecpRawData;	Vector of CRawHeader (See page 218).

3.4.1.18.2 - Constructors

3.4.1.18.2.1 - CSipPacketParser

3.4.1.18.2.1.1 - CSipPacketParser::CSipPacketParser Constructor

Constructor.

C++

```
CSipPacketParser();
```

Description

Constructor. In the initial state, the packet is neither a request or a response.

3.4.1.18.2.1.2 - CSipPacketParser::CSipPacketParser Constructor

Copy constructor.

C++

```
CSipPacketParser( IN const CSipPacketParser& rSrc );
```

Parameters

Parameters	Description
IN const CSipPacketParser& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.18.3 - Destructors

3.4.1.18.3.1 - CSipPacketParser::~CSipPacketParser Destructor

Destructor.

C++

```
virtual ~CSipPacketParser();
```

Description

Destructor.

3.4.1.18.4 - Methods

3.4.1.18.4.1 - CSipPacketParser::AppendRawData Method

Buffers socket-received data. It also tries to parse the start line if not set. Input must be a NULL-terminated string.

C++

```
mxt_result AppendRawData( INOUT const char*& rpcPos );
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the received data. This string must be NULL-terminated. The pointer is adjusted as parsing progresses. Refer to return codes for details on rpcPos's position after this method has completed.

Returns

resSI_SIPPARSER_SIPPACKET_HEADER_INCOMPLETE_START_LINE : Start-line is still incomplete, data buffered for a subsequent call. This is not an error. rpcPos points to the NULL terminator.

resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF : The start line, and all headers, have been found. This packet is considered complete, but may also contain a payload that is not handled by this method. rpcPos points after the double CRLF.

resFE_UNEXPECTED : Error while parsing. Parser expected double CRLF to end *message-header. rpcPos points to the offending character.

Description

This method handles the buffering of the SIP packet's start-line and *message-header. Data received from the network can be buffered by using this method. It may be called multiple times, once each time new data is received on a connection.

This method also buffers each message-header into a CRawHeader (see page 218), and all CRawHeaders are put into a vector that can be retrieved by using the GetRawDataList (see page 316)() method.

The user of this method should append more data until either resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF or an error is

returned.

Once the `resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF` result has been returned, the user is free to use the `GetRawDataList` (see page 316)() and `CommitRawDataList` (see page 315)() methods.

```
RFC 3261 ABNF:
generic-message = start-line
                  *message-header
                  CRLF
                  [ message-body ]
```

See Also

`TryToParseStartLine`, `CRawHeader` (see page 218), `GetRawDataList` (see page 316), `CommitRawDataList` (see page 315), `CHeaderList::AppendRawData` (see page 157).

3.4.1.18.4.2 - CSipPacketParser::CommitRawDataList Method

Instructs `CSipPacketParser` (see page 312) to use its list of buffered raw headers and build its `CHeaderList` (see page 155) with them.

C++

```
mxt_result CommitRawDataList(OUT CVector<GO CSipHeader*>* pvecRefusedRawHeaders = NULL);
```

Parameters

Parameters	Description
<code>pvecRefusedRawHeaders</code>	Pointer to a vector of <code>CSipHeaders</code> . This is an OUT parameter. If this pointer is non-NULL, in the event that a raw header cannot be committed correctly, it is put into this header so that the caller can take action. Ownership of headers put into this vector is given to the caller.

Returns

`resS_OK` : Header list created.

`resSW_SIPPARSER_SIPPACKET_COMMIT_COMPLETE_WITH_APPEND_ERRORS` : Header list created, but errors have been encountered while appending headers. If `pvecRefusedRawHeaders` is non-NULL, check the vector to see which headers caused this warning.

`resFE_SIPPARSER_SIPPACKET_COMMIT_INVALID_STATE` : Either there is no start-line, or there are no headers to commit.

`resFE_INVALID_ARGUMENT` : Error while parsing the status start line.

`resFE_SIPPARSER_STARTLINE_UNKNOWN_PROTOCOL` : Error while parsing the request start line.

`resFE_SIPPARSER_SIPPACKET_INVALID_REQUEST_START_LINE` : Error while parsing the request start line.

Description

This method converts the raw data (vector) list into a parsable `CHeaderList` (see page 155). This method must be called only after the `AppendRawData` (see page 314)() method has returned `resSI_SIPPARSER_SIPPACKET_FOUND_DOUBLE_CRLF`. Once this method has succeeded, the user can use the `CHeaderList` (see page 155) directly. At that time, the raw data list is also removed from memory. This method also automatically parses the start-line (and set the appropriate `m_pRequestLine` or `m_pStatusLine` data member).

See Also

`AppendRawData` (see page 314), `GetRawDataList` (see page 316), `CRawHeader` (see page 218), `CHeaderList` (see page 155), `CHeaderList::GetRawDataList` (see page 162).

3.4.1.18.4.3 - CSipPacketParser::CreateSipMessageBody Method

Creates the message-body, it can be NULL.

C++

```
const CSipMessageBody* CreateSipMessageBody( ) const;
```

Returns

The message-body, can be NULL if there is no Content-Type or if the parsing has failed.

Description

Builds a CSipMessageBody (see page 298) from the payload contained in the packet. All MIME headers found in the SIP packet are duplicated into the CSipMessageBody (see page 298). Once the body is built successfully, it is stored into the packet and can be retrieved by using GetSipMessageBody (see page 317).

This method builds a CSipMessageBody (see page 298) even if the payload is not a "multipart" payload.

Even though this method is const, it internally modifies a data member of the SIP packet. This data member is never used by the SIP stack when receiving a packet, and it is up to the user of the stack to ensure that this data member is set only once, in a thread safe manner, by calling this method

Notes

For a given instance of CSipPacketParser (see page 312), this method will always return the same instance of CSipMessageBody (see page 298)*, if it was previously created, regardless of any changes within the instance.

3.4.1.18.4.4 - GetHeaderList

3.4.1.18.4.4.1 - CSipPacketParser::GetHeaderList Method

Provides access to the header list.

C++

```
const CHeaderList& GetHeaderList() const;
CHeaderList& GetHeaderList();
```

Returns

Reference on the header list.

Description

Provides access to the header list.

3.4.1.18.4.5 - GetPayload

3.4.1.18.4.5.1 - CSipPacketParser::GetPayload Method

Provides access to the optional payload.

C++

```
const CBlob* GetPayload() const;
CBlob* GetPayload();
```

Returns

Constant pointer to the payload. The provided CBlob is NULL-terminated. However, the NULL terminators are not counted in the CBlob's size. The uncounted NULLs allow for text-based parsing, without the need to create a copy of the CBlob whose only purpose would be to add the NULL terminators.

Description

Provides access to the payload.

Warning

If you create a copy of the CBlob, the NULL terminators are NOT added to the resulting copy.

See Also

GetPayload

3.4.1.18.4.6 - CSipPacketParser::GetRawDataList Method

Provides access to the list of buffered headers.

C++

```
mxt_result GetRawDataList(OUT SRawData*& rpstRawData) const;
```

Parameters

Parameters	Description
OUT SRawData*& rpstRawData	Reference to the pointer that is updated if the call succeeds.

Returns

resSW_SIPPARSER_NULL_PTR : There is no raw data list.

resS_OK : There is a raw data list, the pointer has been updated.

Description

Provides access to the raw data list.

See Also

AppendRawData (see page 314), CommitRawDataList (see page 315)

3.4.1.18.4.7 - GetRequestLine**3.4.1.18.4.7.1 - CSipPacketParser::GetRequestLine Method**

Provides access to the request line if available.

C++

```
const CRequestLine* GetRequestLine() const;
CRequestLine* GetRequestLine();
```

Returns

Constant reference to the request line. It can be NULL.

Description

Provides access to the request-line.

See Also

SetRequestLine (see page 320), AppendRawData (see page 314)

3.4.1.18.4.8 - GetSipMessageBody**3.4.1.18.4.8.1 - CSipPacketParser::GetSipMessageBody Method**

Gets the message-body.

Gets the message-body.

C++

```
CSipMessageBody* GetSipMessageBody();
```

Returns

The message-body. It can be NULL.

Description

Gets the message-body associated with this packet.

This method returns NULL until CreateSipMessageBody (see page 315) is called for this instance. After CreateSipMessageBody (see page 315) is called, this method might still return NULL, depending on the content of the packet.

3.4.1.18.4.8.2 - CSipPacketParser::GetSipMessageBody Method

Gets the message-body.

Gets the message-body.

C++

```
const CSipMessageBody* GetSipMessageBody() const;
```

Returns

The message-body. It can be NULL.

Description

Gets the message-body associated with this packet.

This method returns NULL until CreateSipMessageBody (see page 315) is called for this instance. After CreateSipMessageBody (see page 315) is called, this method might still return NULL, depending on the content of the packet.

3.4.1.18.4.9 - GetStatusLine

3.4.1.18.4.9.1 - CSipPacketParser::GetStatusLine Method

Provides access to the status line if available.

C++

```
const CSipStatusLine* GetStatusLine() const;
CSipStatusLine* GetStatusLine();
```

Returns

Constant pointer to the status-line. It can be NULL.

Description

Provides access to the status-line.

See Also

SetStatusLine (see page 320), AppendRawData (see page 314)

3.4.1.18.4.10 - CSipPacketParser::IsLocallyGenerated Method

Returns true if the packet is locally generated.

C++

```
bool IsLocallyGenerated() const;
```

Returns

- true when the packet is locally generated.
- false otherwise.

Description

This method indicates when a packet is locally generated.

3.4.1.18.4.11 - CSipPacketParser::IsRequest Method

Returns true if the packet is a request.

C++

```
bool IsRequest() const;
```

Returns

True if the packet is a request.

Description

Verifies if this packet's start-line is a request-line. A return of false does not necessarily mean that the packet is a response.

3.4.1.18.4.12 - CSipPacketParser::IsResponse Method

Returns true if the packet is a response.

C++

```
bool IsResponse() const;
```

Returns

True if the packet is a response.

Description

Verifies if this packet's start-line is a status-line. A return of false does not necessarily mean that the packet is a request.

3.4.1.18.4.13 - CSipPacketParser::Reset Method

Clears all data back to its initial state.

C++

```
void Reset();
```

Description

Resets the object back to its initial state.

3.4.1.18.4.14 - CSipPacketParser::Serialize Method

Outputs the packet's content into the buffer.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the packet's request or status line, headers, and payload to send on the network.

3.4.1.18.4.15 - CSipPacketParser::SetLocallyGenerated Method

Sets the local packet generation indicator.

C++

```
void SetLocallyGenerated(IN bool bLocallyGenerated);
```

Parameters

Parameters	Description
IN bool bLocallyGenerated	true when the packet is locally generated, false otherwise.

Description

This method allows to set this packet as locally generated.

3.4.1.18.4.16 - CSipPacketParser::SetPayload Method

Sets the optional payload.

C++

```
void SetPayload(IN TO CBlob* pPayload);
```

Parameters

Parameters	Description
IN TO CBlob* pPayload	Payload to set. Ownership is taken.

Description

Sets the payload. If a previous payload is set, it is released and replaced.

See Also

[GetPayload](#) (see page 316)

3.4.1.18.4.17 - CSipPacketParser::SetRequestLine Method

Sets the request line. Converts this packet to a request.

C++

```
void SetRequestLine(IN TO CRequestLine* pRequestLine);
```

Parameters

Parameters	Description
IN TO CRequestLine* pRequestLine	Request line to set. Ownership is taken.

Description

Sets the request line. If a status-line is set, it is released and replaced by the request line. In that case, this method converts the packet from response to request.

See Also

[GetRequestLine](#) (see page 317)

3.4.1.18.4.18 - CSipPacketParser::SetSipMessageBody Method

Sets the message-body.

C++

```
void SetSipMessageBody(IN TO CSipMessageBody* pMessageBody);
```

Parameters

Parameters	Description
IN TO CSipMessageBody* pMessageBody	Message-body to set. Ownership is TAKEN.

Description

Sets the message-body. If a previous payload is set, it is released and replaced.

See Also

[GetSipMessageBody](#) (see page 317)

3.4.1.18.4.19 - CSipPacketParser::SetStatusLine Method

Sets the status line. Converts this packet to a response.

C++

```
void SetStatusLine(IN TO CSipStatusLine* pStatusLine);
```

Parameters

Parameters	Description
IN TO CSipStatusLine* pStatusLine	Status line to set. Ownership is taken.

Description

Sets the status line. If a request-line is set, it is released and replaced by the status line. In that case, this method converts the packet from request to response.

See Also

GetStatusLine (see page 318)

3.4.1.18.5 - Operators**3.4.1.18.5.1 - CSipPacketParser::= Operator**

Assignment operator.

C++

```
CSipPacketParser& operator =(IN const CSipPacketParser& rSrc);
```

Parameters

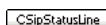
Parameters	Description
IN const CSipPacketParser& rSrc	Source from which to copy.

Returns

A reference on the current instance.

Description

Assignment operator.

3.4.1.19 - CSipStatusLine Class**Class Hierarchy****C++**

```
class CSipStatusLine;
```

Description

The CSipStatusLine class handles the storage, parsing, and serialization of the status-line construct. It is made up of a status code and reason phrase.

The M5T SIP stack does not store the SIP-Version. This stack only supports SIP/2.0 is supported.

```
RFC 3261 ABNF:
Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF
SIP-Version = "SIP" "/" 1*DIGIT "." 1*DIGIT
Status-Code = Informational
/ Redirection
/ Success
/ Client-Error
/ Server-Error
/ Global-Failure
/ extension-code
extension-code = 3DIGIT
Reason-Phrase = *(reserved / unreserved / escaped
/ UTF8-NONASCII / UTF8-CONT / SP / HTAB)
```

Location

SipParser/CSipStatusLine.h

Constructors

Constructor	Description
 CSipStatusLine (see page 322)	Constructor.

Legend

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

Destructors

Destructor	Description
 ~CSipStatusLine (see page 323)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 = (see page 325)	Assignment operator.

Legend

	Method
--	--------

Methods

Method	Description
 GetClass (see page 323)	Returns the status class.
 GetCode (see page 323)	Provides access to the status code.
 GetPhrase (see page 323)	Provides access to the reason phrase.
 Parse (see page 323)	Parses the status-line.
 Reset (see page 324)	Reinitializes data members.
 Serialize (see page 324)	Outputs the contained data into the buffer.
 Set (see page 325)	Sets the status code and reason phrase.

Legend

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

3.4.1.19.1.1 - Constructors

3.4.1.19.1.1 - CSipStatusLine

3.4.1.19.1.1.1 - CSipStatusLine::CSipStatusLine Constructor

Constructor.

C++

```
CSipStatusLine();
```

Description

Constructor.

3.4.1.19.1.1.2 - CSipStatusLine::CSipStatusLine Constructor

Copy constructor.

C++

```
CSipStatusLine(IN const CSipStatusLine& rSrc);
```

Parameters

Parameters	Description
IN const CSipStatusLine& rSrc	Source from which to copy.

Description

Copy constructor. Copies the status code and reason phrase.

3.4.1.19.2 - Destructors**3.4.1.19.2.1 - CSipStatusLine::~CSipStatusLine Destructor**

Destructor.

C++

```
virtual ~CSipStatusLine();
```

Description

Destructor.

3.4.1.19.3 - Methods**3.4.1.19.3.1 - CSipStatusLine::GetClass Method**

Returns the status class.

C++

```
ESipStatusClass GetClass() const;
```

3.4.1.19.3.2 - CSipStatusLine::GetCode Method

Provides access to the status code.

C++

```
uint16_t GetCode() const;
```

3.4.1.19.3.3 - CSipStatusLine::GetPhrase Method

Provides access to the reason phrase.

C++

```
const CString& GetPhrase() const;
```

3.4.1.19.3.4 - CSipStatusLine::Parse Method

Parses the status-line.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. If the method fails, the pointer is set to the position of the error. If the method succeeds, rpcPos points after the CRLF terminating the status-line.

Returns

- resSI_SIPPARSER_DATACONT : Status-line parsed, more data follows.
- resS_OK : Status-line parsed, end of data follows.
- resFE_SIPPARSER_STARTLINE_UNKNOWN_PROTOCOL : SIP-Version is not SIP/2.0.
- resFE_INVALID_ARGUMENT : Could not parse the status code, or status code out-of-range.
- resFE_UNEXPECTED : Could not find terminating CRLF.

Description

Parses the status line. Trailing LWS are not skipped. The terminating CRLF is mandatory. This method validates that the SIP-Version is SIP/2.0. If the version is different, it is not supported by this SIP stack. The reason phrase is not validated syntactically, instead everything following the status code up to the terminating CRLF is considered as the reason phrase. Note that the reason phrase can be empty.

```
RFC 3261 ABNF:
Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF
SIP-Version = "SIP" "/" 1*DIGIT "." 1*DIGIT
Status-Code = Informational
/ Redirection
/ Success
/ Client-Error
/ Server-Error
/ Global-Failure
/ extension-code
extension-code = 3DIGIT
Reason-Phrase = *(reserved / unreserved / escaped
/ UTF8-NONASCII / UTF8-CONT / SP / HTAB)
```

3.4.1.19.3.5 - CSipStatusLine::Reset Method

Reinitializes data members.

C++

```
void Reset();
```

Description

Reinitializes data members. The status code is set to 0 and the reason phrase is empty.

3.4.1.19.3.6 - CSipStatusLine::Serialize Method

Outputs the contained data into the buffer.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the SIP-Version, status code and reason phrase, and terminating CRLF into the buffer. The SIP-Version is hardcoded to "SIP/2.0".

3.4.1.19.3.7 - CSipStatusLine::Set Method

Sets the status code and reason phrase.

C++

```
void Set(IN uint16_t uCode, IN const CString& rstrPhrase = CString());
```

Parameters

Parameters	Description
IN uint16_t uCode	Response code. Valid values are in the 3-digit range of 100 to 699.
IN const CString& rstrPhrase = CString()	Reason phrase. If empty, it is replaced by the default reason phrase if available.

Description

Sets the status code and reason phrase. Sets the default reason phrase if empty.

See Also

GetCode (see page 323), GetPhrase (see page 323)

3.4.1.19.4 - Operators

3.4.1.19.4.1 - CSipStatusLine::= Operator

Assignment operator.

C++

```
CSipStatusLine& operator =(IN const CSipStatusLine& rSrc);
```

Parameters

Parameters	Description
IN const CSipStatusLine& rSrc	Status line from which to initialize.

Returns

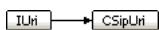
Reference to "this" instance.

Description

Assignment operator. Copies the status code and reason phrase.

3.4.1.20 - CSipUri Class

Class Hierarchy



C++

```
class CSipUri : public IUri;
```

Description

The CSipUri class is used to store, parse, and serialize SIP and SIPS URIs. The SIPS URI form can be set by using the SetSecured (see page 338)() method.

A SIPURI is made up of an optional userinfo section (user and password), a hostport, a parameter list, and an optional header list.

SIPURIs can be escaped. There are four charsets (see CToken (see page 360)) contexts associated with CSipUri. The user, password, parameters, and headers each have different lists characters that are legal and can be escaped. CSipUri handles all manner of escaping automatically.

RFC 3261 ABNF:	SIP-URI = "sip:" [userinfo] hostport uri-parameters [headers]
----------------	--

```

SIPS-URI      = "sips:" [ userinfo ] hostport
userinfo      = ( user / telephone-subscriber ) [ ":" password ] "@"
user          = 1*( unreserved / escaped / user-unreserved )
user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password      = *( unreserved / escaped /
                      "&" / "=" / "+" / "$" / ","
hostport      = host [ ":" port ]
host          = hostname / IPv4address / IPv6reference
hostname      = *( domainlabel ".") toplabel [ "." ]
domainlabel   = alphanum
                  / alphanum *( alphanum / "-" ) alphanum
toplabel       = ALPHA / ALPHA *( alphanum / "-" ) alphanum

IPv4address   = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference = "[" IPv6address "]"
IPv6address   = hexpart [ ":" IPv4address ]
hexpart       = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq        = hex4 *( ":" hex4)
hex4          = 1*4HEXDIG
port          = 1*DIGIT

uri-parameters = *( ";" uri-parameter)
uri-parameter = transport-param / user-param / method-param
                  / ttl-param / maddr-param / lr-param / other-param
transport-param = "transport="
                  ( "udp" / "tcp" / "sctp" / "tls"
                  / other-transport)
other-transport = token
user-param     = "user=" ( "phone" / "ip" / other-user)
other-user     = token
method-param   = "method=" Method
ttl-param      = "ttl=" ttl
maddr-param   = "maddr=" host
lr-param       = "lr"
other-param   = pname [ "=" pvalue ]
pname          = 1*paramchar
pvalue         = 1*paramchar
paramchar      = param-unreserved / unreserved / escaped
param-unreserved = "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers        = "?" header *( "&" header )
header         = hname "=" hvalue
hname          = 1*( hnv-unreserved / unreserved / escaped )
hvalue         = *( hnv-unreserved / unreserved / escaped )
hnv-unreserved = "[" / "]" / "/" / ":" / "&" / "+" / "$"

```

Location

SipParser/CSipUri.h

See Also

CToken (see page 360), CHostPort (see page 167), IUri (see page 380), CGenParamList (see page 146), CGenericParam (see page 140), CHeaderList (see page 155), CSipHeader (see page 239)

Constructors

Constructor	Description
CSipUri (see page 328)	Constructor.

Legend

	Constructor
	Method

Destructors

Destructor	Description
~CSipUri (see page 328)	Destructor.

IUri Class

IUri Class	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 338)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.
 = (see page 339)	Assignment operator.
 == (see page 340)	Comparison operator. Compares as per RFC 3261 sect 19.1.4.

Legend

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

Methods

Method	Description
 GenerateCopy (see page 329)	Generates a copy of this URI.
 GetHeaderList (see page 329)	Provides access to the optional header list.
 GetHostPort (see page 329)	Gets the hostport contained into this URI.
 GetMissingPortBehavior (see page 329)	Gets the missing default port behavior. This behavior is used only when comparing two CSipUri using the IsEquivalent (see page 332) method or the GetParam (see page 330) method.
 GetParam (see page 330)	Gets a header parameter value from the parameter list.
 GetParamList (see page 331)	Provides access to the optional parameter list.
 GetParamTransport (see page 331)	Returns the transport specified in a "transport" parameter.
 GetPassword (see page 331)	Gets the password if present in this URI.
 GetScheme (see page 331)	Returns the URI's scheme. In this case, either SIP or SIPS.
 GetUriType (see page 332)	Returns the URI type, SIP or SIPS.
 GetUser (see page 332)	Gets the user if present in this URI.
 IsEquivalent (see page 332)	Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.
 IsSecured (see page 333)	Returns true if this URI is a "sips".
 Parse (see page 334)	Parses a byte string into useable data.
 Reset (see page 335)	Reinitializes the instance.
 Serialize (see page 335)	Outputs the data member in a format that is ready to send on the network.
 Set (see page 335)	Sets this URI with specified hostname, port, and user. By default, the port is not sent.
 SetHeaderList (see page 335)	Sets the header list.
 SetMissingPortBehavior (see page 336)	Sets the missing default port behavior. This behavior is used only when comparing two CSipUri using the IsEquivalent (see page 332) method or the SetParam (see page 336) method.
 SetParam (see page 336)	Sets a header parameter value to the parameter list from a CString.
 SetParamList (see page 337)	Sets the optional parameter list.
 SetPassword (see page 337)	Sets a password.
 SetSecured (see page 338)	Sets whether or not this URI is "sips".

IUri Class

IUri Class	Description
 GenerateCopy (see page 381)	Generates a copy of this URI.
 GetScheme (see page 382)	Returns the URI's scheme.
 GetUriType (see page 382)	Returns the URI type.
 IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
 Parse (see page 382)	Parses a byte string into usable data.
 Reset (see page 383)	Reinitializes the instance.
 Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	virtual
	abstract

Enumerations

Enumeration	Description
EParameterCreationBehavior (see page 341)	
ESecurityFlag (see page 341)	

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.20.1 - Constructors

3.4.1.20.1.1 - CSipUri

3.4.1.20.1.1.1 - CSipUri::CSipUri Constructor

Constructor.

C++

```
CSipUri();
```

Description

Constructor.

3.4.1.20.1.1.2 - CSipUri::CSipUri Constructor

Copy constructor.

C++

```
CSipUri(IN const CSipUri& rSrc);
```

Parameters

Parameters	Description
IN const CSipUri& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.20.2 - Destructors

3.4.1.20.2.1 - CSipUri::~CSipUri Destructor

Destructor.

C++

```
virtual ~CSipUri();
```

Description

Destructor.

3.4.1.20.3 - Methods

3.4.1.20.3.1 - CSipUri::GenerateCopy Method

Generates a copy of this URI.

C++

```
virtual GO IUuri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.20.3.2 - GetHeaderList

3.4.1.20.3.2.1 - CSipUri::GetHeaderList Method

Provides access to the optional header list.

C++

```
const CHeaderList* GetHeaderList() const;
CHeaderList* GetHeaderList();
```

Returns

Pointer to the header list. It may be NULL.

Description

Provides access to the header list.

3.4.1.20.3.3 - GetHostPort

3.4.1.20.3.3.1 - CSipUri::GetHostPort Method

Gets the hostport contained into this URI.

C++

```
const CHostPort& GetHostPort() const;
CHostPort& GetHostPort();
```

Returns

The host.

Description

Provides access to the host element.

3.4.1.20.3.4 - CSipUri::GetMissingPortBehavior Method

Gets the missing default port behavior. This behavior is used only when comparing two CSipUri (see page 325) using the IsEquivalent (see page 332) method or the

C++

```
static bool GetMissingPortBehavior();
```

Description

Gets the missing default port behavior. This behavior is used only when comparing two CSipUri (see page 325) using the IsEquivalent (see page 332) method or the

True means that if the hostport construct of the URI contains no port, the missing port is considered to be equal to the default port (5060 for a SIP URI and 5061 for a SIPS URI).

False means that if the hostport construct of the URI contains no port, no value is assumed for the missing port.

See Also

[SetMissingPortBehavior](#) (see page 336), [IsEquivalent](#) (see page 332), [operator==](#)

3.4.1.20.3.5 - GetParam

3.4.1.20.3.5.1 - CSipUri::GetParam Method

Gets a header parameter value from the parameter list.

C++

```
const CToken* GetParam(IN const char* pszName) const;
```

Parameters

Parameters	Description
IN const char* pszName	Name of the parameter for which to look. A pre-defined list of parameters exists in SipParser/UriParameter.h.

Returns

A pointer to the value of the requested parameter. It is NULL if it does not exist.

Description

Returns a header parameter value. It can return NULL when the parameter name is not found from the list.

See Also

[GetParamList](#) (see page 331), [SetParamList](#) (see page 337)

3.4.1.20.3.5.2 - CSipUri::GetParam Method

Gets a header parameter value from the parameter list.

C++

```
CToken* GetParam(IN const char* pszName, IN const EParameterCreationBehavior eParameterCreationBehavior = ePARAM_DONT_CREATE);
```

Parameters

Parameters	Description
IN const char* pszName	Name of the parameter for which to look. A pre-defined list of parameters exists in SipParser/UriParameter.h.
IN const EParameterCreationBehavior eParameterCreationBehavior = ePARAM_DONT_CREATE	Enum that specifies the behaviour of this method: <ul style="list-style-type: none"> • ePARAM_DONT_CREATE: A NULL pointer is returned when the pszName is not found from the parameter list. • ePARAM_CREATE_NEW: Add a new parameter to the list when the pszName is not found from the parameter list.

Returns

A pointer to the value of the requested parameter. NULL if it does not exist and ePARAM_DONT_CREATE is specified.

Description

Returns a header parameter value. It can return a NULL when eParameterCreationBehavior is set to ePARAM_DONT_CREATE and the parameter name is not found in the list.

See Also

[GetParamList](#) (see page 331), [SetParamList](#) (see page 337)

3.4.1.20.3.6 - CSipUri::GetParamList

3.4.1.20.3.6.1 - CSipUri::GetParamList Method

Provides access to the optional parameter list.

C++

```
const CGenParamList* GetParamList() const;
CGenParamList* GetParamList();
```

Returns

Pointer to the parameter list. It may be NULL.

Description

Provides access to the parameter list.

3.4.1.20.3.7 - CSipUri::GetParamTransport Method

Returns the transport specified in a "transport" parameter.

C++

```
ESipTransport GetParamTransport() const;
```

Returns

The transport specified in a "transport" parameter.

eINVALID if no "transport" parameter is found or if its value is invalid.

Description

Returns the transport specified in a "transport" parameter.

3.4.1.20.3.8 - CSipUri::GetPassword Method

Gets the password if present in this URI.

C++

```
const CToken* GetPassword() const;
```

Returns

Userinfo's password. It may be NULL.

Description

Provides access to the password.

3.4.1.20.3.9 - CSipUri::GetScheme Method

Returns the URI's scheme. In this case, either SIP or SIPS.

C++

```
virtual const char* GetScheme() const;
```

Returns

URI Scheme.

Description

Returns the scheme, either SIP or SIPS depending on the secured status.

See Also

SetSecured (see page 338)

3.4.1.20.3.10 - CSipUri::GetUriType Method

Returns the URI type, SIP or SIPS.

C++

```
virtual EUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the IsEquivalent (see page 332) method.

3.4.1.20.3.11 - GetUser**3.4.1.20.3.11.1 - CSipUri:: GetUser Method**

Gets the user if present in this URI.

C++

```
CString& GetUser();
const CToken& GetUser() const;
CToken& GetUser();
```

Returns

Userinfo's user part.

Description

Provides access to the username.

3.4.1.20.3.12 - CSipUri::IsEquivalent Method

Compares the given URI with this instance by using applicable RFC rules. For this URI type, compares by using RFC 3261 sect 19.1.4 rules.

C++

```
virtual bool IsEquivalent(IN const IUUri& rSrc) const;
```

Parameters

Parameters	Description
IN const IUUri& rSrc	Source against which to compare.

Returns

True if the URIs are equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

Extract from RFC 3261:
19.1.4 URI Comparison

Some operations in **this** specification require determining whether two SIP or SIPS URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). SIP and SIPS URIs are compared **for** equality according to the following rules:

- o A SIP and SIPS URI are never equivalent.
- o Comparison of the userinfo of SIP and SIPS URIs is **case-sensitive**. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is **case-insensitive** unless explicitly defined otherwise.
- o The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.
- o Characters other than those in the "reserved" set (see RFC 2396 [5]) are equivalent to their "% HEX HEX" encoding.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the user, password, host, and port components must match.

A URI omitting the user component will not match a URI that includes one. A URI omitting the password component will not match a URI that includes one.

A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060. The same is **true for** the transport-parameter, ttl-parameter, user-parameter, and method components.

Defining `sip:user@host` to not be equivalent to `sip:user@host:5060` is a change from RFC 2543. When deriving addresses from URIs, equivalent addresses are expected from equivalent URIs. The URI `sip:user@host:5060` will always resolve to port 5060. The URI `sip:user@host` may resolve to other ports through the DNS SRV mechanisms detailed in [4].
- o URI uri-parameter components are compared as follows:
 - Any uri-parameter appearing in both URIs must match.
 - A user, ttl, or method uri-parameter appearing in only one URI never matches, even if it contains the **default** value.
 - A URI that includes an maddr parameter will not match a URI that contains no maddr parameter.
 - All other uri-parameters appearing in only one URI are ignored when comparing the URIs.
- o URI header components are never ignored. Any present header component **MUST** be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

3.4.1.20.3.13 - CSipUri::IsSecured Method

Returns true if this URI is a "sips".

C++

```
bool IsSecured() const;
```

Returns

True if the URI is SIPS.

Description

Returns the secured status of this URI.

See Also

SetSecured (see page 338)

3.4.1.20.3.14 - CSipUri::Parse Method

Parses a byte string into useable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	A value of IUri::eALLOW_SPECIAL_CHARS indicates that the URI can consider the comma ',', question mark '?', or semi-colon ';' characters as part of this URI. This is as per RFC 3261 conformance item (811) (saying that URIs that contain one of these characters MUST be within angle quotes). This restriction extends to all (name-addr addr-spec) constructs (including SIP extensions). For a SIP or SIPS URI, being within angle quotes allows parsing of trailing parameters and headers.
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. Initially, rpcPos points after the colon following the scheme. If Parse() fails, rpcPos points to the source of the error. URIs do not skip trailing LWS.

Returns

resSI_SIPPARSER_DATACONT : SipUri has been found, more data follows.

resS_OK : SipUri body has been found, end of data follows.

This method can also return any mxt_result that CHostPort::Parse (see page 174) and CGenParamList::Parse (see page 150)() can return, and any error mxt_result that ParseHeaders() may return.

Description

Parses a SIP-URI or SIPS-URI ABNF construct as per RFC 3261, storing the retrieved information in the local data members.

```
RFC 3261 ABNF:
SIP-URI          = "sip:" [ userinfo ] hostport
                  uri-parameters [ headers ]
SIPS-URI         = "sips:" [ userinfo ] hostport
                  uri-parameters [ headers ]
userinfo         = (user / telephone-subscriber) [ ":" password ] "@"
user             = 1*(unreserved / escaped / user-unreserved)
user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password         = *(unreserved / escaped /
                  "&" / "=" / "+" / "$" / ",")
hostport          = host [ ":" port ]
host             = hostname / IPv4address / IPv6reference
hostname         = *(domainlabel ".") toplabel [ "." ]
domainlabel      = alphanum
                  / alphanum *(alphanum / "-") alphanum
toplabel          = ALPHA / ALPHA *(alphanum / "-") alphanum

IPv4address      = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6reference    = "[" IPv6address "]"
IPv6address      = hexpart [ ":" IPv4address ]
hexpart          = hexseq / hexseq ":" [ hexseq ] / ":" [ hexseq ]
hexseq           = hex4 *(": hex4")
hex4             = 1*4HEXDIG
port             = 1*DIGIT

uri-parameters   = *(";" uri-parameter)
uri-parameter    = compression-param / transport-param / user-param
                  / method-param / ttl-param / maddr-param
                  / lr-param / other-param
compression-param = "comp=" ("sigcomp" / other-compression)
other-compression = token
transport-param   = "transport="
                  ("udp" / "tcp" / "sctp" / "tls"
                  / other-transport)
other-transport   = token
```

```

user-param      = "user=" ("phone" / "ip" / other-user)
other-user      = token
method-param    = "method=" Method
ttl-param       = "ttl=" ttl
maddr-param    = "maddr=" host
lr-param        = "lr"
other-param     = pname [ "=" pvalue ]
pname           = 1*paramchar
pvalue          = 1*paramchar
paramchar       = param-unreserved / unreserved / escaped
param-unreserved = "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers         = "?" header *("&" header)
header          = hname "=" hvalue
hname           = 1*(hnv-unreserved / unreserved / escaped)
hvalue          = *(hnv-unreserved / unreserved / escaped)
hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"

```

3.4.1.20.3.15 - CSipUri::Reset Method

Reinitializes the instance.

C++

```
virtual void Reset();
```

Description

Clears all data.

3.4.1.20.3.16 - CSipUri::Serialize Method

Outputs the data member in a format that is ready to send on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the userinfo, host, optional headers, and optional parameters, in the escaped form if necessary.

3.4.1.20.3.17 - CSipUri::Set Method

Sets this URI with specified hostname, port, and user. By default, the port is not sent.

C++

```
void Set(IN const CString& rstrHost, IN uint16_t uPort = CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT, IN
const CString& rstrUser = CString(), IN IN ESecurityFlag eSecured = eUNSECURE);
```

3.4.1.20.3.18 - CSipUri::SetHeaderList Method

Sets the header list.

C++

```
void SetHeaderList(IN TO CHeaderList* pHeaderList);
```

Parameters

Parameters	Description
IN TO CHeaderList* pHeaderList	Header list to set. Ownership is taken. If NULL, clears the header list.

Description

Sets the header list. If a list is already present, it is released and replaced by the new one.

See Also

GetHeaderList (see page 329)

3.4.1.20.3.19 - CSipUri::SetMissingPortBehavior Method

Sets the missing default port behavior. This behavior is used only when comparing two CSipUri (see page 325) using the IsEquivalent (see page 332) method or the

C++

```
static void SetMissingPortBehavior(bool bMissingPortBehavior);
```

Parameters

Parameters	Description
bool bMissingPortBehavior	True if the missing port must be considered equal to the default port. False if the missing port must not be considered equal to the default port.

Description

Sets the missing default port behavior. This behavior is used only when comparing two CSipUri (see page 325) using the IsEquivalent (see page 332) method or the

When set to true, if the hostport construct of the URI contains no port, the missing port is considered to be equal to the default port (5060 for a SIP URI and 5061 for a SIPS URI).

When set to false, if the hostport construct of the URI contains no port, no value is assumed for the missing port.

The value of the port (in the hostport construct) is not changed.

See Also

GetMissingPortBehavior (see page 329), IsEquivalent (see page 332), operator==

3.4.1.20.3.20 - SetParam

3.4.1.20.3.20.1 - CSipUri::SetParam Method

Sets a header parameter value to the parameter list from a CString.

C++

```
void SetParam(IN const char* pszName, IN const CString& rstrValue);
```

Parameters

Parameters	Description
IN const char* pszName	Name of the parameter to set. A pre-defined list of parameters exists in SipParser/UriParameter.h.
IN const CString& rstrValue	Value of the parameter to set.

Description

Sets this URI parameter list with the `pszName` parameter. The parameter is set with the value held by `rstrValue`.

This method takes care of the following:

- If this URI has no parameter list yet, one is created.
- If a `pszName` parameter already exists, its value is replaced with the value held by `rstrValue`.
- If no `pszName` parameter already exists, it is created and appended to the parameter list.

See Also

[GetParamList](#) (see page 331), [SetParamList](#) (see page 337)

3.4.1.20.3.20.2 - CSipUri::SetParam Method

Sets a header parameter value to the parameter list from a CToken (see page 360).

C++

```
void SetParam(IN const char* pszName, IN const CToken& rValue);
```

Parameters

Parameters	Description
IN const char* pszName	Name of the parameter to set. A pre-defined list of parameters exists in SipParser/UriParameter.h.
IN const CToken& rValue	Value of the parameter to set.

Description

Sets this URI parameter list with the `pszName` parameter. The parameter is set with the value held by `rValue`.

This method takes care of the following:

- If this URI has no parameter list yet, one is created.
- If a `pszName` parameter already exists, its value is replaced with the value held by `rValue`.
- If no `pszName` parameter already exists, it is created and appended to the parameter list.

See Also

[GetParamList](#) (see page 331), [SetParamList](#) (see page 337)

3.4.1.20.3.21 - CSipUri::SetParamList Method

Sets the optional parameter list.

C++

```
void SetParamList(IN TO CGenParamList* pParamList);
```

Parameters

Parameters	Description
IN TO CGenParamList* pParamList	Parameter list to set. Ownership is taken. If NULL, clears the parameter list.

Description

Sets the parameter list. If a list is already present, it is released and replaced by the new one.

See Also

[GetParamList](#) (see page 331)

3.4.1.20.3.22 - CSipUri::SetPassword Method

Sets a password.

C++

```
void SetPassword(IN const char* szPassword);
```

Parameters

Parameters	Description
IN const char* szPassword	Pointer to the password. It may be NULL, in which case the current password is cleared.

Description

Sets the password. It is possible to have an empty, non-NULL password. An empty password serializes as "user:@", whereas a NULL password serializes as "user:.".

```
RFC 3261 ABNF:
userinfo      = (user / telephone-subscriber) [ ":" password ] "@"
user          = 1*(unreserved / escaped / user-unreserved)
user-unreserved = "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password      = *(unreserved / escaped /
"&" / "=" / "+" / "$" / "," )
```

See Also

GetPassword (see page 331)

3.4.1.20.3.23 - CSipUri::SetSecured Method

Sets whether or not this URI is "sips".

C++

```
void SetSecured(IN ESecurityFlag eSecured);
```

Parameters

Parameters	Description
IN ESecurityFlag eSecured	If eSECURED, this CSipUri (see page 325) instance is a "sips", otherwise it is a "sip".

Description

Sets the secured state for this URI. Secured means that SIPS URI are used.

See Also

IsSecured (see page 333)

3.4.1.20.4 - Operators

3.4.1.20.4.1 - CSipUri::!= Operator

Comparison operator. Compares as per RFC 3261 sect 19.1.4.

C++

```
bool operator !=(IN const CSipUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CSipUri& rSrc	Source against which to compare.

Returns

True if the URIs are not equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

```
Extract from RFC 3261:
19.1.4 URI Comparison
```

Some operations in **this** specification require determining whether two SIP or SIPS URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). SIP and SIPS URIs are compared **for** equality according to the following rules:

- o A SIP and SIPS URI are never equivalent.

- o Comparison of the userinfo of SIP and SIPS URIs is **case-sensitive**. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is **case-insensitive** unless explicitly defined otherwise.
- o The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.
- o Characters other than those in the "reserved" set (see RFC 2396 [5]) are equivalent to their "% HEX HEX" encoding.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the user, password, host, and port components must match.

A URI omitting the user component will not match a URI that includes one. A URI omitting the password component will not match a URI that includes one.

A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060. The same is **true for** the transport-parameter, ttl-parameter, user-parameter, and method components.

Defining `sip:user@host` to not be equivalent to `sip:user@host:5060` is a change from RFC 2543. When deriving addresses from URIs, equivalent addresses are expected from equivalent URIs. The URI `sip:user@host:5060` will always resolve to port 5060. The URI `sip:user@host` may resolve to other ports through the DNS SRV mechanisms detailed in [4].
- o URI uri-parameter components are compared as follows:
 - Any uri-parameter appearing in both URIs must match.
 - A user, ttl, or method uri-parameter appearing in only one URI never matches, even **if** it contains the **default** value.
 - A URI that includes an maddr parameter will not match a URI that contains no maddr parameter.
 - All other uri-parameters appearing in only one URI are ignored when comparing the URIs.
- o URI header components are never ignored. Any present header component **MUST** be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

See Also

[IsEquivalent](#) (see page 332)

3.4.1.20.4.2 - CSipUri::= Operator

Assignment operator.

C++

```
CSipUri& operator =(IN const CSipUri& rSrc);
```

Parameters

Parameters	Description
IN const CSipUri& rSrc	Source from which to copy.

Returns

Returns a reference on "this" instance.

Description

Assignment operator.

3.4.1.20.4.3 - CSipUri::== Operator

Comparison operator. Compares as per RFC 3261 sect 19.1.4.

C++

```
bool operator ==(IN const CSipUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CSipUri& rSrc	Source against which to compare.

Returns

True if the URIs are equivalent.

Description

Comparison operator. Compares the URIs following RFC 3261 section 19.1.4 rules.

Extract from RFC 3261:
19.1.4 URI Comparison

Some operations in **this** specification require determining whether two SIP or SIPS URIs are equivalent. In **this** specification, registrars need to compare bindings in Contact URIs in REGISTER requests (see Section 10.3.). SIP and SIPS URIs are compared **for** equality according to the following rules:

- o A SIP and SIPS URI are never equivalent.
- o Comparison of the userinfo of SIP and SIPS URIs is **case-sensitive**. This includes userinfo containing passwords or formatted as telephone-subscribers. Comparison of all other components of the URI is **case-insensitive** unless explicitly defined otherwise.
- o The ordering of parameters and header fields is not significant in comparing SIP and SIPS URIs.
- o Characters other than those in the "reserved" set (see RFC 2396 [5]) are equivalent to their "%" HEX HEX encoding.
- o An IP address that is the result of a DNS lookup of a host name does not match that host name.
- o For two URIs to be equal, the user, password, host, and port components must match.

A URI omitting the user component will not match a URI that includes one. A URI omitting the password component will not match a URI that includes one.

A URI omitting any component with a **default** value will not match a URI explicitly containing that component with its **default** value. For instance, a URI omitting the optional port component will not match a URI explicitly declaring port 5060. The same is **true for** the transport-parameter, ttl-parameter, user-parameter, and method components.

Defining `sip:user@host` to not be equivalent to `sip:user@host:5060` is a change from RFC 2543. When deriving addresses from URIs, equivalent addresses are expected from equivalent URIs. The URI `sip:user@host:5060` will always resolve to port 5060. The URI `sip:user@host` may resolve to

other ports through the DNS SRV mechanisms detailed in [4].

- o URI uri-parameter components are compared as follows:
 - Any uri-parameter appearing in both URIs must match.
 - A user, ttl, or method uri-parameter appearing in only one URI never matches, even if it contains the **default** value.
 - A URI that includes an maddr parameter will not match a URI that contains no maddr parameter.
 - All other uri-parameters appearing in only one URI are ignored when comparing the URIs.
- o URI header components are never ignored. Any present header component MUST be present in both URIs and match **for** the URIs to match. The matching rules are defined **for** each header field in Section 20.

See Also

[IsEquivalent](#) (see page 332)

3.4.1.20.5 - Enumerations

3.4.1.20.5.1 - CSipUri::EParameterCreationBehavior Enumeration

```
enum EParameterCreationBehavior {
    ePARAM_DONT_CREATE,
    ePARAM_CREATE_NEW
};
```

Description

Indicates whether or not the parameter must be created when calling [GetParam](#) (see page 330).

Members

Members	Description
ePARAM_DONT_CREATE	Dont' create the parameter.
ePARAM_CREATE_NEW	Create the parameter.

3.4.1.20.5.2 - CSipUri::ESecurityFlag Enumeration

```
enum ESecurityFlag {
    eUNSECURE,
    eSECURE
};
```

Description

Indicates whether or not this URI is "sips".

Members

Members	Description
eUNSECURE	The URI is not secure: "sip"
eSECURE	The URI is secure: "sips"

3.4.1.21 - CStringHelper Class

Class Hierarchy

[CStringHelper](#)

C++

```
class CStringHelper;
```

Location

SipParser/CStringHelper.h

Methods

Method	Description
AdaptForDisplay (see page 343)	Replaces non displayable characters with '!'.
ConvertFromHexAscii (see page 343)	Converts an hexadecimal string into a binary blob.
ConvertToHexAscii (see page 344)	Converts a binary blob into an hexadecimal string.
EscapeChar (see page 344)	Converts from char to "% HEX HEX".
GetEscaped (see page 345)	Converts from "% HEX HEX" to char.
IsAlpha (see page 345)	True if the specified string is made up of ALPHA characters.
IsAlphaNum (see page 345)	True if the specified string is made up of alphanum characters.
IsDigit (see page 346)	True if the specified string is made up of DIGIT characters.
IsFloat (see page 346)	True if the specified string is made up of DIGIT and one DOT characters.
IsHexadecimal (see page 346)	True if the specified string is made up of LHEX or HEXDIG characters.
IsLineTerminator (see page 347)	Returns true if pcPos points to a CRLF with no WSP after.
IsLWS (see page 347)	Returns the number of digits that make up the LWS starting at pcPos.
IsNumeric (see page 348)	True if the specified string is made up of '+' or '-' signs and DIGIT characters.
IsStringQdTextOrQuotedPair (see page 348)	Determines if the given string is made of qdtext or quoted pair.
QuotedStringToString (see page 348)	Converts an RFC 3261 quoted-string to a normal string.
RemoveVisualSeparators (see page 349)	Removes visual separators from a phone number.
SkipLWS (see page 349)	Advances the pointer beyond any LWS up to the next non-LWS character. Returns either resSI_SIIPARSER_DATACONT or resS_OK.
SkipToData (see page 349)	Advances the pointer beyond a specified character, plus any following optional LWS.
SkipWSP (see page 350)	Advances rpcPos over any encountered WSP.
StringToQuotedString (see page 350)	Converts a normal string to an RFC 3261 quoted-string.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
EAllowLws (see page 351)	

3.4.1.21.1 - Data Members

3.4.1.21.1.1 - CStringHelper::ms_cCR Data Member

The carriage return character (0x0A).

C++

```
const char ms_cCR;
```

3.4.1.21.1.2 - CStringHelper::ms_cLF Data Member

The linefeed character (0x0D).

C++

```
const char ms_cLF;
```

3.4.1.21.1.3 - CStringHelper::ms_cNUL Data Member

The NULL character (0x00).

C++

```
const char ms_cNUL;
```

3.4.1.21.1.4 - **CStringHelper::ms_cSP** Data Member

The whitespace character (0x20).

C++

```
const char ms_cSP;
```

3.4.1.21.1.5 - **CStringHelper::ms_cTAB** Data Member

The tabulation character (0x09).

C++

```
const char ms_cTAB;
```

3.4.1.21.1.6 - **CStringHelper::ms_pszCRLF** Data Member

The SIP line terminating pair.

C++

```
const char* const ms_pszCRLF;
```

3.4.1.21.1.7 - **CStringHelper::ms_pszDASHBOUNDARY** Data Member

The dash-boundary pair.

C++

```
const char* const ms_pszDASHBOUNDARY;
```

3.4.1.21.1.8 - **CStringHelper::ms_pszDOUBLECRLF** Data Member

The SIP line double terminating pair.

C++

```
const char* const ms_pszDOUBLECRLF;
```

3.4.1.21.2 - Methods

3.4.1.21.2.1 - **CStringHelper::AdaptForDisplay** Method

Replaces non displayable characters with '.'.

C++

```
static void AdaptForDisplay(INOUT CString& rstrString);
```

Parameters

Parameters	Description
INOUT CString& rstrString	The string to alter.

Description

Replace all non displayable ASCII characters with '.'.

3.4.1.21.2.2 - **CStringHelper::ConvertFromHexAscii** Method

Converts an hexadecimal string into a binary blob.

C++

```
static mxt_result ConvertFromHexAscii(IN const CString& rstrAscii, OUT CBlob& rBlobBinary);
```

Parameters

Parameters	Description
IN const CString& rstrAscii	ID of the proxy.
OUT CBlob& rBlobBinary	Packet to update.

Returns

resS_OK: This string was successfully converted. resFE_FAIL: There are an uneven number of characters or there are non-hexadecimal characters in rstrAscii.

Description

This method converts an hexadecimal string into a binary blob.

3.4.1.21.2.3 - CStringHelper::ConvertToHexAscii Method

Converts a binary blob into an hexadecimal string.

C++

```
static void ConvertToHexAscii(IN const CBlob& rBlobBinary, OUT CString& rstrAscii);
```

Parameters

Parameters	Description
rHash	Blob to convert.
rstrHash	String in which to put the result.

Description

This method converts a binary blob into an hexadecimal string.

3.4.1.21.2.4 - CStringHelper::EscapeChar Method

Converts from char to "% HEX HEX".

C++

```
static void EscapeChar(IN char c, OUT char* pcBuf);
```

Parameters

Parameters	Description
ch	Character to escape.
racBuf	Output buffer where to insert the escaped character. Must be at least 4 characters long (3 chars for escaped form, and one for sprintf's NULL terminator).

Returns

The character in escaped form, including the leading '%'.

Description

Creates the escaped form of 'ch' in a buffer. Note that 'ch' is escaped regardless of its legality in the token's character set. It is up to the user to call EscapeChar only when necessary.

Converts a char into its %HEXHEX equivalent. Note that in RFC 3261, only certain values in the range 0-127 are legal, others must be converted into UTF8 by the application (the stack does not support encoding into UTF8).

<pre>RFC 3261 ABNF escaped = "%" HEXDIG HEXDIG</pre>
<pre>RFC 2234 ABNF HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"</pre>
<p>Note that RFC 2234 states that all strings enclosed in quotation-marks are case-insensitive.</p>

3.4.1.21.2.5 - CStringHelper::GetEscaped Method

Converts from "% HEX HEX" to char.

C++

```
static char GetEscaped(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the character in escaped form. Assumes that IsEscaped has been previously called.

Returns

Returns the character in unescaped form.

Description

Converts a character from escaped to unescaped form.

```
RFC 3261 ABNF
escaped      = "%" HEXDIG HEXDIG

RFC 2234 ABNF
HEXDIG       = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

Note that RFC 2234 states that all strings enclosed in
quotation-marks are case-insensitive.
```

See Also

EscapeChar (see page 344)

3.4.1.21.2.6 - CStringHelper::IsAlpha Method

True if the specified string is made up of ALPHA characters.

C++

```
static bool IsAlpha(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

True if the string is made up of alphabetical characters only.

Description

Checks whether or not the provided string is only made up of alphabetical characters (a-z, A-Z).

3.4.1.21.2.7 - CStringHelper::IsAlphaNum Method

True if the specified string is made up of alphanum characters.

C++

```
static bool IsAlphaNum(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

True if the string is made up of alphanumeric characters only.

Description

Checks whether or not the provided string is only made up of alphanumeric characters (a-z, A-Z, 0-9).

3.4.1.21.2.8 - CStringHelper::IsDigit Method

True if the specified string is made up of DIGIT characters.

C++

```
static bool IsDigit(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

True if the string is made up of digit characters only.

Description

Checks whether or not the provided string is only made up of alphanumeric characters (0-9).

3.4.1.21.2.9 - CStringHelper::IsFloat Method

True if the specified string is made up of DIGIT and one DOT characters.

C++

```
static bool IsFloat(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

True if the string is a float.

Description

Checks if the provided string is a float. The dot is mandatory; however, the dot's prefix can be empty. A floating-point must be decimal.

Example

```
"1234567890.1" is a floating-point string.
".2" is a floating-point string.
"1234567890" is not a floating-point string.
"1." is not a floating-point string.
"1234567890A.1" is not a floating-point string.
```

3.4.1.21.2.10 - CStringHelper::IsHexadecimal Method

True if the specified string is made up of LHEX or HEXDIG characters.

C++

```
static bool IsHexadecimal(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

Returns true if the token is hexadecimal only. An empty string is considered as not being hexadecimal.

Description

Checks whether or not the string is made up exclusively of hexadecimal characters.

Example

<pre>"abcdeF1234567890" is an hexadecimal string. "abcdeFG1234567890" is not an hexadecimal string.</pre>

3.4.1.21.2.11 - CStringHelper::IsLineTerminator Method

Returns true if pcPos points to a CRLF with no WSP after.

C++

```
static bool IsLineTerminator(IN const char* pcPos);
```

Parameters

Parameters	Description
IN const char* pcPos	Pointer to the data to verify. May or may not be NULL-terminated.

Returns

Returns true if pcPos points to a single CRLF with no WSP after.

Description

Verifies if pcPos points to the start of a line terminator. RFC 3261 only allows "CRLF" as a line terminator. Also note that CRLF followed by LWS is considered as LWS, not as a line terminator. See RFC 3261 section 28.1 pg 257.

See Also

[IsLWS](#) (see page 347)

3.4.1.21.2.12 - CStringHelper::IsLWS Method

Returns the number of digits that make up the LWS starting at pcPos.

C++

```
static unsigned int IsLWS(IN const char* pcPos);
```

Parameters

Parameters	Description
IN const char* pcPos	Start position. May or may not be NULL-terminated.

Returns

Returns the number of characters that make up the LWS pointed by szString. Can return 0 if LWS structure is not found.

Description

Determines if the pointer points to a LWS structure, and if so finds out how many characters make it up. Note that a white space is mandatory, and that a CRLF followed by WSP is considered as LWS, not as a line terminator.

<pre>RFC 3261 ABNF LWS = [*WSP CRLF] 1*WSP ; linear whitespace SWS = [LWS] ; sep whitespace RFC 2234 ABNF WSP = SP / HTAB</pre>
--

See Also

[IsLineTerminator](#) (see page 347), [SkipWSP](#) (see page 350), [SkipLWS](#) (see page 349)

3.4.1.21.2.13 - CStringHelper::IsNumeric Method

True if the specified string is made up of '+' or '-' signs and DIGIT characters.

C++

```
static bool IsNumeric(IN const char* szString);
```

Parameters

Parameters	Description
IN const char* szString	NULL-terminating string to verify.

Returns

Returns true if the token is numeric. An empty string is considered as not being numeric.

Description

Checks whether or not the string is made up exclusively of numeric characters.

Example

```
"1234567890" is a numeric string.  
"1234567890A" is not a numeric string.
```

3.4.1.21.2.14 - CStringHelper::IsStringQdTextOrQuotedPair Method

Determines if the given string is made of qdtext or quoted pair.

C++

```
static bool IsStringQdTextOrQuotedPair(IN const CString& rStr);
```

Parameters

Parameters	Description
IN const CString& rStr	The string to verify.

Returns

True if the string is either a Qd Text or a quoted pair.

Description

This method verifies if the first character of the given string is a qdtext or a quoted pair.

See Also

[IsQdText](#), [IsQuotedPair](#)

3.4.1.21.2.15 - CStringHelper::QuotedStringToString Method

Converts an RFC 3261 quoted-string to a normal string.

C++

```
static mxt_result QuotedStringToString(INOUT CString& rStr);
```

Parameters

Parameters	Description
INOUT CString& rStr	[IN] The string to modify. [OUT] If the method returns resS_OK, the modified string. If the method returns resFE_INVALID_ARGUMENT, the original string.

Returns

- resS_OK, on success.
- resFE_INVALID_ARGUMENT, if the string contains illegal characters for quoted strings or if the string is not a quoted string.

Description

Transforms an RFC 3261 quoted string to a normal string. This method removes the quotes around the quoted string and also properly un-escapes the reserved characters in order for them to be properly displayed.

This method leaves all UTF-8 characters as is.

For the quoted string BNF, see [StringToQuotedString](#) (see page 350).

See Also

[StringToQuotedString](#) (see page 350)

3.4.1.21.2.16 - **CStringHelper::RemoveVisualSeparators** Method

Removes visual separators from a phone number.

C++

```
static void RemoveVisualSeparators(INOUT CString& rstrPhoneNumber);
```

Parameters

Parameters	Description
INOUT CString& rstrPhoneNumber	Phone number to strip.

Description

Removes all visual separators as described in RFC 3966 Section 3.

global-number-digits = "+" *phonedigit DIGIT *phonedigit local-number-digits = *phonedigit-hex (HEXDIG / "*" / "#")*phonedigit-hex
 phonedigit = DIGIT / [visual-separator] phonedigit-hex = HEXDIG / "*" / "#" / [visual-separator] visual-separator = "-" / "." / "(" / ")"

3.4.1.21.2.17 - **CStringHelper::SkipLWS** Method

Advances the pointer beyond any LWS up to the next non-LWS character. Returns either [resSI_SIPPARSER_DATACONT](#) or [resS_OK](#).

C++

```
static mxt_result SkipLWS(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Position from which to start looking.

Returns

[resSI_SIPPARSER_DATACONT](#) : Data continues after the LWS. LWS may or may not have been found.

[resS_OK](#) : Data ends after the LWS. LWS may or may not have been found.

Description

Advances `rpcPos` over any encountered LWS. This method considers that the LWS is optional and does not return an error in any case. If LWS is mandatory, users must use the [IsLWS](#) (see page 347)() method.

See Also

[IsLWS](#) (see page 347), [UpdateReturn](#)

3.4.1.21.2.18 - **CStringHelper::SkipToData** Method

Advances the pointer beyond a specified character, plus any following optional LWS.

C++

```
static mxt_result SkipToData(INOUT const char*& rpcPos, IN char cSkip);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Position from which to start looking.
IN char cSkip	Mandatory character to find at rpcPos.

Returns

resSI_SIPPARSER_DATACONT : Character found, data continues after the optional LWS.
 resS_OK : Character found, data ends after the optional LWS.
 resFE_SIPPARSER_STRHELP_CHAR_NOT_FOUND : Character not found, processing ended there. LWS has not been skipped.

Description

Checks for a specified character, and if it finds it, advances rpcPos beyond it and any LWS following it. Note that this method processes the LWS as optional. If LWS is not optional, manually skip over the character and use IsLWS (see page 347)().

See Also

IsLWS (see page 347)

3.4.1.21.2.19 - CStringHelper::SkipWSP Method

Advances rpcPos over any encountered WSP.

C++

```
static unsigned int SkipWSP(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Position where to start skipping. It is updated while skipping occurs.

Returns

Number of characters skipped.

Description

Advances rpcPos over any encountered WSP.

RFC 2234 ABNF
WSP = SP / HTAB

3.4.1.21.2.20 - CStringHelper::StringToQuotedString Method

Converts a normal string to an RFC 3261 quoted-string.

C++

```
static mxt_result StringToQuotedString(INOUT CString& rStr);
```

Parameters

Parameters	Description
INOUT CString& rStr	[IN] The string to modify. [OUT] If the method returns resS_OK, the modified string. If the method returns resFE_INVALID_ARGUMENT, the original string.

Returns

- resS_OK, on success.
- resFE_INVALID_ARGUMENT, if the string contains illegal characters for quoted strings.

Description

Transforms the string received in parameter to follow the RFC 3261 quoted string construct. This method adds quotes around the string and properly escapes the reserved characters.

Stack users should always use this method when configuring quoted strings in the stack.

This method leaves all UTF-8 characters as is.

quoted string BNF:

```

RFC 2234
WSP      =  SP / HTAB
DQUOTE  =  %x22      ; " (Double Quote)

RFC 3261
LWS      =  [*WSP CRLF] 1*WSP ; linear whitespace
SWS      =  [LWS]           ; sep whitespace
quoted-string =  SWS DQUOTE *(qdtext / quoted-pair ) DQUOTE
qdtext   =  LWS / %x21 / %x23-5B / %x5D-7E / UTF8-NONASCII
quoted-pair =  "" (%x00-09 / %x0B-0C / %x0E-7F)

```

See Also

QuotedStringToString (see page 348)

Example

As an example, a user wants to display his name "Max" using slash and backslash characters (i.e., //ax). This method properly escapes the backslash character, since it is illegal within a quoted string.

```

CString strDisplayName = "/\\/\\"ax";
CNameAddr nameAddr;

CStringHelper::StringToQuotedString(strDisplayName)
nameAddr.SetSipUri("example.com",
                  CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT,
                  "max",
                  CSipUri::eUNSECURE,
                  strDisplayName);

```

The display name returned by nameAddr.GetDisplayName() then looks like in the SIP packet:

"/\\/\\"ax"

3.4.1.21.3 - Enumerations

3.4.1.21.3.1 - CStringHelper::EAllowLws Enumeration

```

enum EAllowLws {
    eNO_LWS,
    eALLOW_LWS
};

```

Description

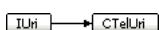
Indicate whether or not LWS is allowed. See ConditionalSkipLWS.

Members

Members	Description
eNO_LWS	LWS is not allowed when calling ConditionalSkipLWS.
eALLOW_LWS	LWS is allowed when calling ConditionalSkipLWS.

3.4.1.22 - CTelUri Class

Class Hierarchy



C++

```
class CTelUri : public IUuri;
```

Description

The CTelUri class is used to store, parse, and serialize tel URI.

A tel URI is made up of a global-number or local-number and a parameter list. A local-number MUST include a phone-context parameter.

```
RFC 3966:
telephone-uri      = "tel:" telephone-subscriber
telephone-subscriber = global-number / local-number
global-number        = global-number-digits *par
local-number         = local-number-digits *par context *par
par                 = parameter / extension / isdn-subaddress
isdn-subaddress     = ";isub=" 1*uric
extension           = ";ext=" 1*phonedigit
context              = ";phone-context=" descriptor
descriptor           = domainname / global-number-digits
global-number-digits = "+" *phonedigit DIGIT *phonedigit
local-number-digits  =
    *phonedigit-hex (HEXDIG / "*" / "#")*phonedigit-hex
domainname           = *( domainlabel "." ) toplabel [ "." ]
domainlabel          = alphanum
    / alphanum *( alphanum / "-" ) alphanum
    / ALPHA / ALPHA *( alphanum / "-" ) alphanum
    = ";" pname [= pvalue ]
    = 1*( alphanum / "-" )
    = 1*paramchar
    = param-unreserved / unreserved / pct-encoded
    = alphanum / mark
    = "-" / "_" / "." / "!" / "~" / "*" /
    "!" / "(" / ")"
    = "%" HEXDIG HEXDIG
    = "[" / "]" / "/" / ":" / "&" / "+" / "$"
    = DIGIT / [ visual-separator ]
    = HEXDIG / "*" / "#" / [ visual-separator ]
    = "-" / "." / "(" / ")"
    = ALPHA / DIGIT
    = ";" / "/" / "?" / ":" / "@" / "&" /
    "=" / "+" / "$" / ","
    = reserved / unreserved / pct-encoded
uric                = reserved / unreserved / pct-encoded
```

Warning

The 'isdn-subaddress' parameter (isub=) is not fully supported in the actual implementation of the CTelUri. This is caused by the 'reserved' characters that can not be stored in the CToken (see page 360) of the CGenericParam (see page 140). The 'isdn-subaddress' parameter may only be used by filling it with standard SipUri parameter characters. Moreover, the actual implementation can not guarantee that the 'isdn-subaddress' or 'extension' (ext=) parameters appear first before the 'context' (phone-context=) parameter. See RFC 3966 Section 3 (URI Syntax) for more information.

Location

SipParser/CTelUri.h

See Also

IUri (see page 380), CSipUri (see page 325), CAbsoluteUri (see page 128), CToken (see page 360), CHostPort (see page 167), CGenParamList (see page 146), CGenericParam (see page 140)

Constructors

Constructor	Description
CTelUri (see page 354)	Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
~CTelUri (see page 354)	Destructor.

IUri Class

IUri Class	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 359)	Comparison operator. Compares as per RFC 3966 section 4.
 = (see page 360)	Assignment operator.
 == (see page 360)	Comparison operator. Compares as per RFC 3966 section 4.

Legend

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

Methods

Method	Description
  GenerateCopy (see page 354)	Generates a copy of this URI.
  GetParamList (see page 355)	Provides access to the optional parameter list.
  GetPhoneContext (see page 355)	Gets the phone context. (Only when IsGlobalPhoneNumber (see page 356)() == false)
  GetPhoneNumber (see page 355)	Gets the phone number.
  GetScheme (see page 355)	Returns the URI's scheme. In this case TEL.
  GetUriType (see page 356)	Returns the URI type, TEL.
  IsEquivalent (see page 356)	Compares the given URI with this instance using applicable RFC rules. For this URI type, compares as per RFC3966 section 4 rules.
  IsGlobalPhoneNumber (see page 356)	Returns true if the telephone-subscriber is a global phone number.
  Parse (see page 357)	Parses a byte string into useable data.
  Reset (see page 358)	Reinitializes the instance.
  Serialize (see page 358)	Outputs the data member in a format that is ready to be sent on the network.
  SetGlobalNumber (see page 358)	Sets this URI with specified global phone number.
  SetLocalNumber (see page 358)	Sets this URI with specified local phone number and phone context.
  SetParamList (see page 359)	Sets the optional parameter list.

IUri Class

IUri Class	Description
  GenerateCopy (see page 381)	Generates a copy of this URI.
  GetScheme (see page 382)	Returns the URI's scheme.
  GetUriType (see page 382)	Returns the URI type.
  IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
  Parse (see page 382)	Parses a byte string into usable data.
  Reset (see page 383)	Reinitializes the instance.
  Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	virtual
	abstract

Enumerations

IUri Class

IUri Class	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.22.1 - Constructors

3.4.1.22.1.1 - CTelUri

3.4.1.22.1.1.1 - CTelUri::CTelUri Constructor

Constructor.

C++

```
CTelUri();
```

Description

Constructor.

3.4.1.22.1.1.2 - CTelUri::CTelUri Constructor

Copy constructor.

C++

```
CTelUri(IN const CTelUri& rSrc);
```

Parameters

Parameters	Description
IN const CTelUri& rSrc	Source from which to copy.

Description

Copy constructor.

3.4.1.22.2 - Destructors

3.4.1.22.2.1 - CTelUri::~CTelUri Destructor

Destructor.

C++

```
virtual ~CTelUri();
```

Description

Destructor.

3.4.1.22.3 - Methods

3.4.1.22.3.1 - CTelUri::GenerateCopy Method

Generates a copy of this URI.

C++

```
virtual GO IUuri* GenerateCopy() const;
```

Returns

A copy of this instance. Ownership of the copy is given to the caller.

Description

Creates a copy of this instance and gives ownership to the caller.

3.4.1.22.3.2 - GetParamList

3.4.1.22.3.2.1 - CTelUri::GetParamList Method

Provides access to the optional parameter list.

C++

```
const CGenParamList* GetParamList() const;
CGenParamList* GetParamList();
```

Returns

Pointer to the parameter list. It may be NULL.

Description

Provides access to the parameter list.

3.4.1.22.3.3 - GetPhoneContext

3.4.1.22.3.3.1 - CTelUri::GetPhoneContext Method

Gets the phone context. (Only when IsGlobalPhoneNumber (see page 356)() == false)

C++

```
const CToken* GetPhoneContext() const;
CToken* GetPhoneContext();
```

Returns

Pointer to the local phone numbers context. It may return NULL.

Description

Provides access to the local phone numbers context.

3.4.1.22.3.4 - GetPhoneNumber

3.4.1.22.3.4.1 - CTelUri::GetPhoneNumber Method

Gets the phone number.

C++

```
const CToken* GetPhoneNumber() const;
CToken* GetPhoneNumber();
```

Returns

Pointer to the Phone number.

Description

Provides access to the telephone-subscriber phone number.

3.4.1.22.3.5 - CTelUri::GetScheme Method

Returns the URI's scheme. In this case TEL.

C++

```
virtual const char* GetScheme() const;
```

Returns

URI Scheme.

Description

Returns the scheme TEL.

3.4.1.22.3.6 - CTelUri::GetUriType Method

Returns the URI type, TEL.

C++

```
virtual EUriType GetUriType() const;
```

Returns

URI type.

Description

Returns the TEL type.

3.4.1.22.3.7 - CTelUri::IsEquivalent Method

Compares the given URI with this instance using applicable RFC rules. For this URI type, compares as per RFC3966 section 4 rules.

C++

```
virtual bool IsEquivalent(IN const IUri& rSrc) const;
```

Parameters

Parameters	Description
IN const IUri& rSrc	Source against which to compare.

Returns

True if the URIs are equivalent.

Description

Comparison operator. Compares the URIs following RFC 3966 section 4 rules.

Extract from RFC 3966:

4. URI Comparisons

Two "tel" URIs are equivalent according to the following rules:

- o Both must be either a 'local-number' or a 'global-number', i.e., start with a '+'.
- o The 'global-number-digits' and the 'local-number-digits' must be equal, after removing all visual separators.
- o For mandatory additional parameters (section 5.4) and the 'phone-context' and 'extension' parameters defined in **this** document, the 'phone-context' parameter value is compared as a host name **if** it is a 'domainname' or digit by digit **if** it is 'global-number-digits'. The latter is compared after removing all 'visual-separator' characters.
- o Parameters are compared according to 'pname', regardless of the order they appeared in the URI. If one URI has a parameter name not found in the other, the two URIs are not equal.
- o URI comparisons are **case-insensitive**.

All parameter names and values **SHOULD** use lower-**case** characters, as tel URIs may be used within contexts where comparisons are **case-sensitive**.

Section 19.1.4 in the SIP specification [RFC3261] discusses one such **case**.

3.4.1.22.3.8 - CTelUri::IsGlobalPhoneNumber Method

Returns true if the telephone-subscriber is a global phone number.

C++

```
bool IsGlobalPhoneNumber() const;
```

Returns

True when the telephone-subscriber is a global phone number.

Description

Checks if the telephone-subscriber is a global phone number.

3.4.1.22.3.9 - CTelUri::Parse Method

Parses a byte string into useable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	Not used.
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. Initially, rpcPos points after the colon following the scheme. If Parse() fails, rpcPos points to the source of the error. URLs do not skip trailing LWS.

Returns

resSI_SIPPARSER_DATACONT : TelUri is found, more data follows.

resSW_SIPPARSER_TELURI_PHONE_CONTEXT_NOT_FOUND : The phone-context parameter is not found. It is mandatory for local phone numbers.

resS_OK : TelUri body is found, end of data follows.

This method can also return any mxt_result that CGenParamList::Parse (see page 150)() can return.

Description

Parses a TEL-URI as per RFC 3966.

```
RFC 3966:
telephone-uri      = "tel:" telephone-subscriber
telephone-subscriber = global-number / local-number
global-number        = global-number-digits *par
local-number         = local-number-digits *par context *par
par                 = parameter / extension / isdn-subaddress
isdn-subaddress     = ";isub=" 1*uric
extension           = ";ext=" 1*phonedigit
context              = ";phone-context=" descriptor
descriptor           = domainname / global-number-digits
global-number-digits = "+" *phonedigit DIGIT *phonedigit
local-number-digits = *
*phonedigit-hex (HEXDIG / "*" / "#")*phonedigit-hex
domainname           = *(domainlabel ".") toplabel [ ". " ]
domainlabel          = alphanum
                     / alphanum *(alphanum / "-") alphanum
toplabel              = ALPHA / ALPHA *(alphanum / "-") alphanum
parameter             = ";" pname [= pvalue ]
pname                = 1*(alphanum / "-")
pvalue                = 1*paramchar
paramchar              = param-unreserved / unreserved / pct-encoded
unreserved             = alphanum / mark
mark                 = "-" / "_" / "." / "!" / "~" / "*" /
                     " " / "(" / ")"
pct-encoded            = "%" HEXDIG HEXDIG
param-unreserved      = "[" / "]" / "/" / ":" / "&" / "+" / "$"
phonedigit            = DIGIT / [ visual-separator ]
phonedigit-hex        = HEXDIG / "*" / "#" / [ visual-separator ]
visual-separator      = "-" / "." / "(" / ")"
alphanum              = ALPHA / DIGIT
reserved               = ";" / "/" / "?" / ":" / "@" / "&" /
```

uric	"=" / "+" / "\$" / ",," = reserved / unreserved / pct-encoded
------	--

3.4.1.22.3.10 - CTelUri::Reset Method

Reinitializes the instance.

C++

```
virtual void Reset();
```

Description

Clears all data.

3.4.1.22.3.11 - CTelUri::Serialize Method

Outputs the data member in a format that is ready to be sent on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Output buffer.

Description

Outputs the phone number, phone context, and optional parameters, in the escaped form if necessary.

3.4.1.22.3.12 - CTelUri::SetGlobalNumber Method

Sets this URI with specified global phone number.

C++

```
mxt_result SetGlobalNumber(IN const CString& rstrPhoneNumber);
```

Parameters

Parameters	Description
IN const CString& rstrPhoneNumber	Global phone number to set. The '+' is added when not provided.

Returns

resS_OK : TelUri is set.

resSI_SIPPARSER_DATACONT: Phone number is found. Not all of the data has been parsed.

resFE_SIPPARSER_TOKEN_NOT_FOUND: rstrPhoneNumber contains invalid char.

resFE_INVALID_ARGUMENT: rstrPhoneNumber is empty.

Description

Populates the minimal value to have a valid global TelUri.

3.4.1.22.3.13 - CTelUri::SetLocalNumber Method

Sets this URI with specified local phone number and phone context.

C++

```
mxt_result SetLocalNumber(IN const CString& rstrPhoneNumber, IN const CString& rstrPhoneContext);
```

Parameters

Parameters	Description
IN const CString& rstrPhoneNumber	Local phone number to set.
IN const CString& rstrPhoneContext	Local phone context to set.

Returns

resS_OK : TelUri is set.

resSI_SIPPARSER_DATACONT: Phone number is found. Not all of the data has been parsed.

resFE_SIPPARSER_TOKEN_NOT_FOUND: rstrPhoneNumber contains invalid char.

resFE_INVALID_ARGUMENT: One of the following cause:

- the phone context is not provided or
- the phone number is not provided or
- the phone number is a global phone number.

Description

Populates the minimal value to have a valid local TelUri.

3.4.1.22.3.14 - CTelUri::SetParamList Method

Sets the optional parameter list.

C++

```
void SetParamList(IN TO CGenParamList* pParamList);
```

Parameters

Parameters	Description
IN TO CGenParamList* pParamList	Parameter list to set. Ownership is taken. If NULL, clears the parameter list.

Description

Sets the parameter list. If a list is already present, it is released and replaced by the new one.

Warning

If the TelUri is a local phone number, the parameter list contains the MANDATORY parameter 'phone-context' as per RFC 3966 Section 5.1.5.. Using SetParamList may remove this mandatory parameter and CAUSE THE TELURI TO BE INVALID.

See Also

GetParamList (see page 355)

3.4.1.22.4 - Operators

3.4.1.22.4.1 - CTelUri::!= Operator

Comparison operator. Compares as per RFC 3966 section 4.

C++

```
bool operator !=(IN const CTelUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CTelUri& rSrc	Source against which to compare.

Returns

True if the URIs are not equivalent.

Description

Comparison operator.

See Also

operator==, IsEquivalent (see page 356)

3.4.1.22.4.2 - CTelUri::= Operator

Assignment operator.

C++

```
CTelUri& operator =(IN const CTelUri& rSrc);
```

Parameters

Parameters	Description
IN const CTelUri& rSrc	Source from which to copy.

Returns

Returns a reference on "this" instance.

Description

Assignment operator.

3.4.1.22.4.3 - CTelUri::== Operator

Comparison operator. Compares as per RFC 3966 section 4.

C++

```
bool operator ==(IN const CTelUri& rSrc) const;
```

Parameters

Parameters	Description
IN const CTelUri& rSrc	Source against which to compare.

Returns

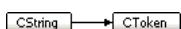
True if the URIs are equivalent.

Description

Comparison operator.

See Also

operator!=, IsEquivalent (see page 356)

3.4.1.23 - CToken Class**Class Hierarchy****C++**

```
class CToken : protected CString;
```

Description

This class handles parsing and storage for the basic token structure. The token is the building block of almost all RFC 3261 ABNF. A token is like a word. Spaces delimit tokens.

RFC 3261 declares a few tokens. One is used inside headers, others are used within the SipUri. Each token has a different character set, meaning that some characters are valid and others are not in the context of a token. Refer to the ms_aTokenData array for a list of

each token's character set.

Some tokens are case sensitive, and some can contain escaped values (such as an escaped 'space' character: %20). This is also hardcoded in the ms_aTokenData table.

This class implements all token types. The ECharSet (see page 378) tells CToken how to handle parsing and serialization by giving it access to the correct structure in the ms_aTokenData. Through it, the CToken knows which characters are legal, which should be escaped, and how comparison should be made (case sensitive or not). The charset is set at the CToken construction, and will never change.

Special note: Upon Serialization, CToken automatically handles escaping. Unescapable tokens are serialized as they are -- with no further validation. It is thus possible for the user to set a token string that does not follow RFC 3261 syntax for tokens that are not escapable.

Location

SipParser/CToken.h

Constructors

Constructor	Description
CToken (see page 362)	Constructor from a CString.

Legend

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

Destructors

Destructor	Description
 ~CToken (see page 363)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
 != (see page 373)	Comparison operator.
 < (see page 374)	Comparison operator.
 = (see page 375)	Assignment Operator.
 == (see page 377)	Comparison operator.

Legend

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

Methods

Method	Description
 FindTokenEnd (see page 364)	Returns a pointer to the first character that is not part of the charset.
 GetCharSet (see page 364)	Returns the charset for which this token is configured.
 GetFloat (see page 364)	Provides access to conversion of a float / double string to an unsigned integer and a multiplier.
 GetInt32 (see page 365)	Provides conversion from string to number.
 GetParserGrammar (see page 365)	Gets whether or not the specified element in the given character set is allowed.
 GetString (see page 366)	Provides access to the token data.
 GetUint16 (see page 366)	
 GetUint32 (see page 365)	Provides conversion from string to number.
 GetUint64 (see page 366)	Provides conversion from string to number.
 IsAlpha (see page 366)	Returns true if the token is alphabetic only.
 IsEmpty (see page 367)	Returns true if the token is empty.
 IsFloat (see page 367)	Returns true if the token is a floating-point only.
 IsHexadecimal (see page 367)	Returns true if the token is hexadecimal only.

◆ IsInt32 (See page 367)	Returns true if the token is an unsigned integer from -2,147,483,647 to +2,147,483,647. The sign is optional.
◆ IsLwsAllowed (See page 368)	Returns true if the token's context (based on the charset) allows whitespace.
◆ IsNumeric (See page 368)	Returns true if the token is numeric only.
◆ IsSignedInteger (See page 369)	Returns true if the token is a signed integer.
◆ IsUInt16 (See page 369)	Returns true if the token is an unsigned integer from 0 to 65535.
◆ IsUInt32 (See page 370)	Returns true if the token is an unsigned integer from 0 to 4,294,967,295.
◆ IsUInt64 (See page 370)	Returns true if the token is an unsigned integer from 0 to +18,446,744,073,709,551,615.
◆ IsUnsignedInteger (See page 370)	Returns true if the token is an unsigned integer.
◆ IsValidChar (See page 371)	Returns true if 'ch' is part of the charset supported by this type of token.
◆ Length (See page 371)	Returns the token's length.
◆ Parse (See page 372)	Parses a token up to the first character that is not part of the charset. Also skips LWS.
◆ Reset (See page 372)	Clears the token.
◆ Serialize (See page 372)	Outputs the token. Outputs escaped characters if the token type supports it.
◆ SetFloat (See page 372)	Sets a float value into the token.
◆ SetParserGrammar (See page 373)	Updates the grammar of a given character set.

Legend

◆	Method
---	--------

Enumerations

Enumeration	Description
ECharSet (See page 378)	Supported charsets

3.4.1.23.1 - Constructors

3.4.1.23.1.1 - CToken

3.4.1.23.1.1.1 - CToken::CToken Constructor

Constructor from a CString.

C++

```
CToken( IN ECharSet eCharSet, IN const CString& rstrSrc );
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set to use. This affects case comparison, characters that are considered legal, and escaping.
IN const CString& rstrSrc	The string to assign to this instance.

Description

Extended constructor.

Warning

Note that this constructor does not verify that the assigned string is only made up of legal characters for the current token type. Care must be taken when using token types that cannot be escaped.

3.4.1.23.1.1.2 - CToken::CToken Constructor

Constructor from a c-style string.

C++

```
CToken( IN ECharSet eCharSet, IN const char* szSrc = NULL );
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set to use. This affects case comparison, characters that are considered legal, and escaping.
IN const char* szSrc = NULL	The token to set. It can be NULL.

Description

Extended constructor.

Warning

Note that this constructor does not verify that the assigned string is only made up of legal characters for the current token type. Care must be taken when using token types that cannot be escaped.

3.4.1.23.1.1.3 - CToken::CToken Constructor

Constructor from a c-style string and length.

C++

```
CToken(IN ECharSet eCharSet, IN const char* szSrc, IN uint16_t uCount);
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set to use. This affects case comparison, characters that are considered legal, and escaping.
IN const char* szSrc	The token to set. It should not be NULL.
IN uint16_t uCount	Number of bytes to include in this token.

Description

Extended constructor.

Warning

Note that this constructor does not verify that the assigned string is only made up of legal characters for the current token type. Care must be taken when using token types that cannot be escaped.

3.4.1.23.1.1.4 - CToken::CToken Constructor

Copy constructor.

C++

```
CToken(IN const CToken& rSrc);
```

Parameters

Parameters	Description
IN const CToken& rSrc	Token to copy.

Description

Extended constructor.

3.4.1.23.2 - Destructors**3.4.1.23.2.1 - CToken::~CToken Destructor**

Destructor.

C++

```
virtual ~CToken();
```

Description

Destructor.

3.4.1.23.3 - Methods**3.4.1.23.3.1 - CToken::FindTokenEnd Method**

Returns a pointer to the first character that is not part of the charset.

C++

```
const char* FindTokenEnd(IN const char* pcStart) const;
```

Parameters

Parameters	Description
IN const char* pcStart	Token's start position.

Returns

Returns a pointer to the first character that is not part of the charset.

Description

Finds the end of a token by checking whether or not the characters are part of the charset.

3.4.1.23.3.2 - CToken::GetCharSet Method

Returns the charset for which this token is configured.

C++

```
ECharSet GetCharSet() const;
```

Returns

The character set for which this token has been created.

Description

Provides access to the character set that this token implements. The charset is specified at construction time, and will never change throughout the life of the token.

3.4.1.23.3.3 - CToken::GetFloat Method

Provides access to conversion of a float / double string to an unsigned integer and a multiplier.

C++

```
mxt_result GetFloat(OUT int& ruValue, OUT unsigned int& ruMultiplicator) const;
```

Parameters

Parameters	Description
OUT unsigned int& ruMultiplicator	The value by which ruValue is multiplied to get an integer. The multiplicator is always a positive integer equaling a power of 10.
rnValue	Converted value of the token. Note that the float value that read is multiplied by ruMultiplicator to get the integer ruValue. The value is negative if the number begins with a '-'.

Returns

resS_OK : Conversion is successful.

resFE_FAIL: The token is of another type.

Description

Provides access to the conversion of a float / double string to an unsigned integer and a multiplier.

This method parses strings that are numbers as told by IsNumeric (see page 368).

The OUT parameter rnValue is set to zero and ruMultiplicator is set to one when this function returns resFE_FAIL. The user of this function must not rely on rnValue and ruMultiplicator having these values to detect an error, as "0.0" still yields the above output, with a result of resS_OK.

See Also

IsNumeric (see page 368)

Example

```
"12345" will result in ruValue = 12345 and ruMultiplicator = 1.
"12345.1" will result in ruValue = 12345 and ruMultiplicator = 10.
"1234A.1" will fail and result in ruValue = 0 and ruMultiplicator = 1.
```

3.4.1.23.3.4 - CToken::GetInt32 Method

Provides conversion from string to number.

C++

```
mxt_result GetInt32(OUT int32_t& rnValue) const;
mxt_result GetUint32(OUT uint32_t& ruValue) const;
```

Parameters

Parameters	Description
OUT int32_t& rnValue	Converted value of the token. 0 if unsuccessful.
OUT uint32_t& ruValue	Converted value of the token. 0 if unsuccessful.

Returns

resS_OK : Conversion is successful.

resFE_INVALID_ARGUMENT: The token is of another type.

Description

Provides access to the conversion from string to int32_t.

3.4.1.23.3.5 - CToken::GetParserGrammar Method

Gets whether or not the specified element in the given character set is allowed.

C++

```
static mxt_result GetParserGrammar(IN const unsigned int uIndex, IN const ECharSet eCharSet, OUT bool& rbValue);
```

Parameters

Parameters	Description
IN const unsigned int uIndex	Index of the value to get in eCharSet.
IN const ECharSet eCharSet	Character set in which to get a grammar value.
bValue	Boolean specifying if the specified element is allowed or not.

Returns

- resS_OK upon success an error otherwise.

Description

Gets whether or not a the specified element in the given character set is allowed.

3.4.1.23.3.6 - GetString

3.4.1.23.3.6.1 - CToken::GetString Method

Provides access to the token data.

C++

```
CString& GetString();
const CString& GetString() const;
```

Returns

The contained token.

Description

Provides access to the token data.

3.4.1.23.3.7 - CToken::GetUInt16 Method

```
mxt_result GetUInt16(OUT uint16_t& ruValue) const;
mxt_result GetUInt16(OUT uint16_t& ruValue) const;
```

Parameters

Parameters	Description
OUT uint16_t& ruValue	Converted value of the token. 0 if unsuccessful.
OUT uint16_t& ruValue	Converted value of the token. 0 if unsuccessful.

Returns

resS_OK : Conversion is successful.

resFE_INVALID_ARGUMENT: The token is of another type.

Description

Provides access to the conversion from string to uint16_t.

3.4.1.23.3.8 - CToken::GetUInt64 Method

Provides conversion from string to number.

C++

```
mxt_result GetUInt64(OUT uint64_t& ruValue) const;
```

3.4.1.23.3.9 - CToken::IsAlpha Method

Returns true if the token is alphabetic only.

C++

```
bool IsAlpha() const;
```

Returns

Returns true if the token is alphabetic only. An empty string is considered as not being alphabetic.

Description

Checks whether or not the token is made up exclusively of alphabetic characters.

See Also

CStringHelper::IsAlpha (see page 345)

Example

```
"abcdeFRD" is an alphabetic string.
"abc3deFRD" is not an alphabetic string.
```

3.4.1.23.3.10 - CToken::IsEmpty Method

Returns true if the token is empty.

C++

```
bool IsEmpty() const;
```

Returns

True if the token is empty.

Description

Returns true if the token is empty.

3.4.1.23.3.11 - CToken::IsFloat Method

Returns true if the token is a floating-point only.

C++

```
bool IsFloat() const;
```

Returns

Returns true if the token is a floating-point. An empty string is considered as not being floating-point.

Description

Checks whether or not the token is made up exclusively of floating-point characters.

See Also

[CStringHelper::IsFloat](#) (see page 346)

Example

```
"1234567890" is NOT a floating-point string.  
"1234567890.1" is a floating-point string.  
"1234567890A.1" is NOT a floating-point string.
```

3.4.1.23.3.12 - CToken::IsHexadecimal Method

Returns true if the token is hexadecimal only.

C++

```
bool IsHexadecimal() const;
```

Returns

Returns true if the token is hexadecimal only. An empty string is considered as not being hexadecimal.

Description

Checks whether or not the token is made up exclusively of hexadecimal characters.

See Also

[CStringHelper::IsHexadecimal](#) (see page 346)

Example

```
"abcdeF1234567890" is an hexadecimal string.  
"abcdeFG1234567890" is not an hexadecimal string.
```

3.4.1.23.3.13 - CToken::IsInt32 Method

Returns true if the token is an unsigned integer from -2,147,483,647 to +2,147,483,647. The sign is optional.

C++

```
bool IsInt32() const;
```

Returns

True if the token is a signed integer between -2,147,483,647 and +2,147,483,647 inclusively.

Description

Checks whether or not the token is a 4 byte signed int.

See Also

IsInt32, IsSignedInteger (see page 369)

Example

```
"-2147483647" is a 32 bit signed integer string.  
"+2147483647" is a 32 bit signed integer string.  
"2147483647" is a 32 bit signed integer string.  
"+-2147483647" is not a 32 bit signed integer string.  
"12345678900000546547987" is not a 32 bit signed integer string.  
"AAA123" is not a 32 bit signed integer string.
```

3.4.1.23.3.14 - IsLwsAllowed**3.4.1.23.3.14.1 - CToken::IsLwsAllowed Method**

Returns true if the token's context (based on the charset) allows whitespace.

C++

```
CStringHelper::EAllowLws IsLwsAllowed() const;
```

Returns

Returns CStringHelper::eALLOW_LWS if the token's context (based on the charset) allows whitespace.

Description

LWS is allowed only within SIP Headers and SIP Header Parameters.

3.4.1.23.3.14.2 - CToken::IsLwsAllowed Method

Returns true if the token's context (based on the charset) allows whitespace.

C++

```
static CStringHelper::EAllowLws IsLwsAllowed(IN ECharSet eCharSet);
```

Parameters

Parameters	Description
IN ECharSet eCharSet	Character set to check.

Returns

Returns CStringHelper::eALLOW_LWS if the token's context (based on the charset) allows whitespace.

Description

LWS is allowed only within SIP Headers and SIP Header Parameters.

3.4.1.23.3.15 - CToken::IsNumeric Method

Returns true if the token is numeric only.

C++

```
bool IsNumeric() const;
```

Returns

Returns true if the token is numeric. An empty string is considered as not being numeric.

Description

Checks whether or not the token is made up exclusively of numeric characters.

See Also

[CStringHelper::IsNumeric](#) (see page 348)

Example

```
"1234567890" is a numeric string.  
"1234567890A" is not a numeric string.
```

3.4.1.23.3.16 - CToken::IsSignedInteger Method

Returns true if the token is a signed integer.

C++

```
bool IsSignedInteger() const;
```

Returns

Returns true if the token is a signed integer. An empty string is considered as not being a signed integer.

Description

Checks if the token is made up exclusively of unsigned integer characters (digits).

See Also

[CStringHelper::IsDigit](#) (see page 346)

Example

```
"1234567890" is a signed integer string.  
"-1234567890" is a signed integer string.  
"+1234567890" is a signed integer string.  
"" is a not signed integer string.  
"1234567890A" is not a signed integer string.  
"--1234567890A" is not a signed integer string.
```

3.4.1.23.3.17 - CToken::IsUint16 Method

Returns true if the token is an unsigned integer from 0 to 65535.

C++

```
bool IsUint16() const;
```

Returns

True if the token is an unsigned between 0 and 65535 inclusively.

Description

Checks whether or not the token is a 2 byte unsigned int.

See Also

[IsUint32](#) (see page 370), [IsUnsignedInteger](#) (see page 370)

Example

```
"65535" is a 16 bit unsigned integer string.  
"-4000" is not a 16 bit unsigned integer string.  
"+4000" is not a 16 bit unsigned integer string.  
"1234567890" is not a 16 bit unsigned integer string.
```

3.4.1.23.3.18 - CToken::IsUInt32 Method

Returns true if the token is an unsigned integer from 0 to 4,294,967,295.

C++

```
bool IsUInt32() const;
```

Returns

True if the token is an unsigned between 0 and 4,294,967,295 inclusively.

Description

Checks whether or not the token is a 4 byte unsigned int.

See Also

IsUInt16 (see page 369), IsUnsignedInteger (see page 370)

Example

```
"4294967295" is a 32 bit unsigned integer string.  
"-4000" is not a 32 bit unsigned integer string.  
"+4000" is not a 32 bit unsigned integer string.  
"12345678900000546547987" is not a 32 bit unsigned integer string.
```

3.4.1.23.3.19 - CToken::IsUInt64 Method

Returns true if the token is an unsigned integer from 0 to +18,446,744,073,709,551,615.

C++

```
bool IsUInt64() const;
```

Returns

True if the token is a unsigned integer between 0 to +18,446,744,073,709,551,615 inclusively.

Description

Checks if the token is a 8 byte unsigned int.

3.4.1.23.3.20 - IsUnsignedInteger

3.4.1.23.3.20.1 - CToken::IsUnsignedInteger Method

Returns true if the token is an unsigned integer.

C++

```
bool IsUnsignedInteger() const;
```

Returns

Returns true if the token is an unsigned integer. An empty string is considered as not being an unsigned integer.

Description

Checks whether or not the token is made up exclusively of unsigned integer characters (digits).

See Also

CStringHelper::IsDigit (see page 346)

Example

```
"1234567890" is an unsigned integer string.  
"-1234567890" is not an unsigned integer string.  
"+1234567890" is not an unsigned integer string.  
"1234567890A" is not an unsigned integer string.
```

3.4.1.23.3.20.2 - CToken::IsUnsignedInteger Method

Returns true if the token is an unsigned integer.

C++

```
bool IsUnsignedInteger(IN const char* szString) const;
```

Parameters

Parameters	Description
IN const char* szString	String to check.

Returns

Returns true if the token is an unsigned integer. An empty string is considered as not being an unsigned integer.

Description

Checks whether or not the token is made up exclusively of unsigned integer characters (digits).

See Also

CStringHelper::IsDigit (see page 346)

Example

```
"1234567890" is an unsigned integer string.  
"-1234567890" is not an unsigned integer string.  
"+1234567890" is not an unsigned integer string.  
"1234567890A" is not an unsigned integer string.
```

3.4.1.23.3.21 - CToken::IsValidChar Method

Returns true if 'ch' is part of the charset supported by this type of token.

C++

```
bool IsValidChar(IN char c) const;
```

Parameters

Parameters	Description
ch	Character to validate.

Returns

True if 'ch' is part of the character set supported by this token.

Description

This method provides a way to verify if a certain character is legal for the token charset.

3.4.1.23.3.22 - CToken::Length Method

Returns the token's length.

C++

```
uint16_t Length() const;
```

Returns

Returns the length of the contained token.

Description

Provides the length of the contained token.

3.4.1.23.3.23 - CToken::Parse Method

Parses a token up to the first character that is not part of the charset. Also skips LWS.

C++

```
mxt_result Parse(INOUT const char*& rpcPos);
```

Parameters

Parameters	Description
INOUT const char*& rpcPos	Pointer to the data to parse. It is adjusted as parsing progresses. CToken (see page 360) automatically advances rpcPos over any LWS encountered AFTER parsing its data.

Returns

resSI_SIPPARSER_DATACONT : Token found, optional LWS skipped (only applies when IsLwsAllowed (see page 368)), more data follows.

resS_OK : Token found, optional LWS skipped (only applies when IsLwsAllowed (see page 368)), end of data follows.

resFE_SIPPARSER_TOKEN_NOT_FOUND : Token not found, processing ended there. LWS not skipped. Certain token types such as SipUri tokens are valid even when empty.

Description

Parses a single token, advancing rpcPos as it goes. All characters that are part of the current character set are copied into the CToken (see page 360). Characters are automatically unescaped if the type of token supports it. The parsed data is kept in human-readable unescaped form.

The trailing LWS is skipped, but only if CToken (see page 360) can infer from its charset that LWS is allowed. If LWS is not allowed, rpcPos is set to the end of the token.

3.4.1.23.3.24 - CToken::Reset Method

Clears the token.

C++

```
void Reset();
```

Description

Clears the internal data (the token string). Does not affect the character set.

3.4.1.23.3.25 - CToken::Serialize Method

Outputs the token. Outputs escaped characters if the token type supports it.

C++

```
void Serialize(INOUT CBlob& rBlob) const;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	Where to output the internal data.

Description

Outputs the CToken (see page 360)'s data. If the token type supports escaping, illegal characters are escaped before being output. If the token type does not support escaping, the token contents is output as-is, with no validity checks.

3.4.1.23.3.26 - CToken::SetFloat Method

Sets a float value into the token.

C++

```
mxt_result SetFloat(IN int nFloatValue, IN unsigned int uFloatMultiplier);
```

Parameters

Parameters	Description
IN int nFloatValue	The value of the float multiplied by uFloatMultiplier to make an integer.
IN unsigned int uFloatMultiplier	The multiplier used to make the nFloatValue integer value from the float value. MUST be a non-0 power of 10 (1, 10, 100, 1000, etc.).

Returns

resFE_INVALID_ARGUMENT: uFloatMultiplier is invalid. The token is left unchanged in this case.

resS_OK: the value has been set into the token.

Description

Sets the token with a float value. Only copies the string, does not modify the charset.

3.4.1.23.3.27 - CToken::SetParserGrammar Method

Updates the grammar of a given character set.

C++

```
static mxt_result SetParserGrammar(IN const unsigned int uIndex, IN const bool bValue, IN const ECharSet eCharSet);
```

Parameters

Parameters	Description
IN const unsigned int uIndex	Index of the value to change in eCharSet.
IN const bool bValue	Whether or not the character at the specified index in the specified character set is legal.
IN const ECharSet eCharSet	Character set in which to change a grammar value.

Returns

- resS_OK upon success.
- resFE_INVALID_ARGUMENT if eCharSet or uIndex is invalid.

Description

Changes whether or not an element is legal in a specified character set. An element's index is the same as in an ASCII table. For more information on the current character set please refer to RFC 3261 section 25.1.

Warning

Changing the grammar in any way can have important side effects and may potentially cause crashes. M5T should be consulted before doing so.

Notes:

Must be called before ISipCoreConfig::Startup (see page 66).

3.4.1.23.4 - Operators**3.4.1.23.4.1 - !=****3.4.1.23.4.1.1 - CToken::!= Operator**

Comparison operator.

C++

```
bool operator !=(IN const CString& rstrSrc) const;
```

Parameters

Parameters	Description
IN const CString& rstrSrc	CString against which to test.

Returns

True if the internal string is different from rstrSrc's string.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the charset configuration provided at the constructor.

3.4.1.23.4.1.2 - CToken::!= Operator

Comparison operator.

C++

```
bool operator !=(IN const CToken& rSrc) const;
```

Parameters

Parameters	Description
IN const CToken& rSrc	Token against which to test.

Returns

True if the internal string is different from rSrc's string.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the charset configuration provided at the constructor.

3.4.1.23.4.1.3 - CToken::!= Operator

Comparison operator.

C++

```
bool operator !=(IN const char* szSrc) const;
```

Parameters

Parameters	Description
IN const char* szSrc	String against which to test.

Returns

True if the internal string is different from szSrc's string.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the charset configuration provided at the constructor.

3.4.1.23.4.2 - CToken::< Operator

Comparison operator.

C++

```
bool operator <(IN const CToken& rSrc) const;
```

Parameters

Parameters	Description
IN const CToken& rSrc	Token against which to test.

Returns

True if the character set of this instance is (numerically) "less than" the one of rSrc. False if it's "greater than". If the character sets are equal, the internal strings are compared, case-sensitively or not depending on the character set itself.

Description

Comparison operator. The string comparison is either case sensitive or not, depending on the character set provided at construction. This operation doesn't make sense in itself, but is meant to be used in maps and similar structures where a CToken (see page 360) can be a key in a (key, value) pair. Indeed, the semantics of this operation are appropriate in this context, where it is rational to say that a token is "less than" another without referring strictly to the internal string but to the token as a whole. Hence the inclusion of the character set in the comparison.

3.4.1.23.4.3 - =**3.4.1.23.4.3.1 - CToken::= Operator**

Assignment Operator.

C++

```
CToken& operator =(IN const CString& rstrSrc);
```

Parameters

Parameters	Description
IN const CString& rstrSrc	CString to copy.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

Warning

Note that this operator does not verify that the assigned string is only made up of legal characters for the current token type. Care must be taken when using token types that cannot be escaped.

3.4.1.23.4.3.2 - CToken::= Operator

Assignment Operator.

C++

```
CToken& operator =(IN const CToken& rSrc);
```

Parameters

Parameters	Description
IN const CToken& rSrc	CToken (see page 360) to copy.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

Warning

Note that this operator does not verify that the assigned string is only made up of legal characters for the current token type. Care must

be taken when using token types that cannot be escaped.

3.4.1.23.4.3.3 - CToken::= Operator

Assignment Operator.

C++

```
CToken& operator =(IN const char* szSrc);
```

Parameters

Parameters	Description
IN const char* szSrc	String to copy. It should not be NULL.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

Warning

Note that this operator does not verify that the assigned string is only made up of legal characters for the current token type. Care must be taken when using token types that cannot be escaped.

3.4.1.23.4.3.4 - CToken::= Operator

Assignment Operator.

C++

```
CToken& operator =(IN int32_t nValue);
```

Parameters

Parameters	Description
IN int32_t nValue	Integer to copy. Internally, the data is stored in string form.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

3.4.1.23.4.3.5 - CToken::= Operator

Assignment Operator.

C++

```
CToken& operator =(IN uint32_t uValue);
```

Parameters

Parameters	Description
IN uint32_t uValue	Unsigned integer to copy. Internally, the data is stored in string form.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

3.4.1.23.4.3.6 - CToken::= Operator

Assignment Operator.

C++

```
CToken& operator =(IN uint64_t uValue);
```

Parameters

Parameters	Description
IN uint64_t uValue	Unsigned integer to copy. Internally, the data is stored in string form.

Returns

Returns a reference on this instance.

Description

Assignment operator. Only copies the string, does not modify the charset.

3.4.1.23.4.4 - ==

3.4.1.23.4.4.1 - CToken::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CString& rstrSrc) const;
```

Parameters

Parameters	Description
IN const CString& rstrSrc	CString against which to test.

Returns

True if the internal string is equal to rstrSrc's string.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the charset configuration provided at the constructor.

3.4.1.23.4.4.2 - CToken::== Operator

Comparison operator.

C++

```
bool operator ==(IN const CToken& rSrc) const;
```

Parameters

Parameters	Description
IN const CToken& rSrc	Token against which to test.

Returns

True if the internal string is equal to rSrc's string and the token type is the same.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the eCharSet configuration provided at the constructor.

3.4.1.23.4.4.3 - CToken::== Operator

Comparison operator.

C++

```
bool operator ==(IN const char* szSrc) const;
```

Parameters

Parameters	Description
IN const char* szSrc	String against which to test.

Returns

True if the internal string is equal to szSrc's string. A NULL pointer is never equal to anything.

Description

Comparison operator. Comparison is either case sensitive or not, depending on the charset configuration provided at the constructor.

3.4.1.23.5 - Enumerations**3.4.1.23.5.1 - CToken::ECharSet Enumeration**

Supported charsets

C++

```
enum ECharSet {
    eCS_SIP_HEADER = 0,
    eCS_SIPURI_USER,
    eCS_SIPURI_PASSWORD,
    eCS_SIPURI_PARAM,
    eCS_SIPURI_HEADER,
    eCS_TELURI_TELEPHONE_SUBSCRIBER,
    eCS_TELURI_PARAM,
    eCS_SIPHEADER_PARAM,
    eCS_LAST
};
```

Members

Members	Description
eCS_SIP_HEADER = 0	Tokens within a Sip Header context.
eCS_SIPURI_USER	Tokens within a SipUri user name context.
eCS_SIPURI_PASSWORD	Tokens within a SipUri password context.
eCS_SIPURI_PARAM	Tokens within a SipUri parameter context.
eCS_SIPURI_HEADER	Tokens within a SipUri header context.
eCS_TELURI_TELEPHONE_SUBSCRIBER	Tokens within a TelUri telephone subscriber context.
eCS_TELURI_PARAM	Tokens within a TelUri parameter context.
eCS_SIPHEADER_PARAM	Tokens within a SIP Header Parameter.
eCS_LAST	Keep last.

3.4.1.24 - CURiFactory Class**Class Hierarchy**

```
CUriFactory
```

C++

```
class CUriFactory;
```

Description

This class has the responsibility of instanciating the correct type of URI when parsing. It looks at the scheme and then creates the correct parser.

It is also responsible to parse the URI once its scheme has been found, and also handles the bracket enclosing setting.

IUri (see page 380) users MUST use this class to parse and serialize URIs. This is required since it enables easy extension and

support of new URI types.

Location

SipParser/CUriFactory.h

See Also

CSipUri (see page 325), CAbsoluteUri (see page 128), IUri (see page 380)

Methods

Method	Description
◆ CompareScheme (see page 379)	Compares the scheme using the rule stated in RFC 2396 (case-insensitive).
◆ ParseUri (see page 379)	Parses a URI, including the brackets, the scheme, and the body.

Legend

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

3.4.1.24.1 - Methods

3.4.1.24.1.1 - CUriFactory::CompareScheme Method

Compares the scheme using the rule stated in RFC 2396 (case-insensitive).

C++

```
static bool CompareScheme(IN const char* szFirstScheme, IN const char* szSecScheme);
```

Parameters

Parameters	Description
IN const char* szFirstScheme	First scheme to compare.
IN const char* szSecScheme	Second scheme against which to compare.

Returns

True if the schemes are the same.

Description

Compares two scheme strings, case-insensitively as per RFC 2396 section 3.1.

3.4.1.24.1.2 - CUriFactory::ParseUri Method

Parses a URI, including the brackets, the scheme, and the body.

C++

```
static mxt_result ParseUri(IN IUri::ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos, OUT GO IUri*& rpUri);
```

Parameters

Parameters	Description
IN IUri::ESpecialCharactersAllowed eAllowSpecials	Set to EALLOW_SPECIAL_CHARS to indicate that the URI can consider trailing parameters and headers as being part of the URI.
INOUT const char*& rpcPos	Pointer to the start of the URI. Initially points either on an opening angle bracket '<' or to the start of the scheme. It is adjusted while parsing progresses. Trailing LWS is not skipped after the URI. If ParseUri() fails, rpcPos points to the offending sequence.
OUT GO IUri*& rpUri	IUri (see page 380) that has been instantiated for parsing. If ParseUri() succeeds, the scheme is set in IUri (see page 380). If ParseUri() fails, rpUri is set to NULL. Ownership is given to caller.

Returns

resSI_SIPPARSER_DATACONT : URI found, more data follows.

resS_OK : URI found, end of data follows.

resFE_UNEXPECTED : No URI body after the scheme could be found.

resFE_INVALID_ARGUMENT : Scheme does not start with an alpha character.

*Can also return any code that a IUri::Parse (see page 382)() implementor can return.

Description

Parses a complete URI, including scheme and body. This method does *NOT* handle angle quotes. Once the scheme has been found, the appropriate type of IUri (see page 380) is instantiated and is used to parse the body.

Note that LWS is not accepted within URIs and this method does not skip trailing LWS.

This method does NOT create URIs without scheme (e.g. CMailboxUri (see page 182)).

See Also

CSipUri::Parse (see page 334), CAbsoluteUri::Parse (see page 132), ParseScheme

3.4.1.25 - IUri Class

Class Hierarchy

 IUri

C++

`class IUri;`

Description

The IUri is the base class for all URI types that are supported by the SIP stack. It contains basic operations that all URI-abstracting classes should implement.

The IUri interface is used in the SIP stack everywhere a URI can be used. For example, the request-line would be abstracted with a IUri.

```
Uses of IUri in the SIP stack:
RFC3261 ABNF:
Request-URI = SIP-URI / SIPS-URI / TEL-URI / absoluteURI /
IM-URI / PRES-URI

Alert-Info = "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param = LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Call-Info = "Call-Info" HCOLON info *(COMMA info)
info = LAQUOT absoluteURI RAQUOT *( SEMI info-param )

name-addr = [ display-name ] LAQUOT addr-spec RAQUOT
addr-spec = SIP-URI / SIPS-URI / TEL-URI / absoluteURI

Error-Info = "Error-Info" HCOLON error-uri *(COMMA error-uri)
error-uri = LAQUOT absoluteURI RAQUOT *( SEMI generic-param )

Proxy-Authenticate = "Proxy-Authenticate" HCOLON challenge
challenge = ("Digest" LWS digest-cln *(COMMA digest-cln))
            / other-challenge
other-challenge = auth-scheme LWS auth-param
                  *(COMMA auth-param)
digest-cln = realm / domain / nonce
            / opaque / stale / algorithm
            / qop-options / auth-param
realm = "realm" EQUAL realm-value
realm-value = quoted-string
domain = "domain" EQUAL LDQUOT URI
            *( 1*SP URI ) RDQUOT
URI = absoluteURI / abs-path
```

Location

SipParser/IUri.h

See Also

CUriFactory (see page 378)

Destructors

Destructor	Description
~IUri (see page 381)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
GenerateCopy (see page 381)	Generates a copy of this URI.
GetScheme (see page 382)	Returns the URI's scheme.
GetUriType (see page 382)	Returns the URI type.
IsEquivalent (see page 382)	Compares the given URI with this instance by using applicable RFC rules.
Parse (see page 382)	Parses a byte string into usable data.
Reset (see page 383)	Reinitializes the instance.
Serialize (see page 383)	Outputs the data members in a format that is ready to be sent on the network.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
ESpecialCharactersAllowed (see page 383)	Enum to use in parameter to the IUri::Parse (see page 382) method.
EUriType (see page 384)	All URI types must be listed in this enum.

3.4.1.25.1 - Destructors

3.4.1.25.1.1 - IUri::~IUri Destructor

Destructor.

C++

```
virtual ~IUri();
```

3.4.1.25.2 - Methods

3.4.1.25.2.1 - IUri::GenerateCopy Method

Generates a copy of this URI.

C++

```
virtual GO IUri* GenerateCopy() const = 0;
```

Returns

The new IUri (see page 380) child class copy. Ownership of this instance is given to the caller.

Description

Allocates a new instance of the current URI type and copies its content into it. The use of this method is that users of the IUri (see page 380) class are able to create copies of their URIs by using a pointer to the base class (IUri (see page 380)) rather than having to typecast it.

3.4.1.25.2.2 - **IUri::GetScheme** Method

Returns the URI's scheme.

C++

```
virtual const char* GetScheme() const = 0;
```

Returns

URI's scheme.

Description

Returns the URI's scheme.

See Also

SetScheme

3.4.1.25.2.3 - **IUri::GetUriType** Method

Returns the URI type.

C++

```
virtual EUriType GetUriType() const = 0;
```

Returns

URI type.

Description

Returns the URI type. This method is required because of the IsEquivalent (see page 382) method.

3.4.1.25.2.4 - **IUri::IsEquivalent** Method

Compares the given URI with this instance by using applicable RFC rules.

C++

```
virtual bool IsEquivalent(IN const IUri& rSrc) const = 0;
```

Parameters

Parameters	Description
IN const IUri& rSrc	The URI against which to verify the equivalence.

Returns

Returns true if rSrc is an equivalent URI following comparison methods underlined in RFC 3261 or the appropriate RFC for the implementing URI type.

Description

Compares the given URI with this instance by using applicable RFC rules.

3.4.1.25.2.5 - **IUri::Parse** Method

Parses a byte string into usable data.

C++

```
virtual mxt_result Parse(IN ESpecialCharactersAllowed eAllowSpecials, INOUT const char*& rpcPos) = 0;
```

Parameters

Parameters	Description
IN ESpecialCharactersAllowed eAllowSpecials	A value of IUri::eALLOW_SPECIAL_CHARS indicates that the URI can consider the comma ',', question mark '?', or semi-colon ';' characters as part of this URI. This is as per RFC 3261 conformance item {811} (saying that URIs that contain one of these characters MUST be within angle quotes). This restriction extends to all (name-addr addr-spec) constructs (including SIP extensions).
INOUT const char*& rpcPos	Position from which to start. rpcPos initially points after the scheme and colon. rpcPos is adjusted as parsing progresses. If Parse() fails, rpcPos points to the position that caused the failure. URIs MUST NOT skip trailing LWS.

Returns

See the child implementation for details.

Description

Parses a URI.

3.4.1.25.2.6 - IUri::Reset Method

Reinitializes the instance.

C++

```
virtual void Reset() = 0;
```

Description

Reinitializes the instance.

3.4.1.25.2.7 - IUri::Serialize Method

Outputs the data members in a format that is ready to be sent on the network.

C++

```
virtual void Serialize(INOUT CBlob& rBlob) const = 0;
```

Parameters

Parameters	Description
INOUT CBlob& rBlob	CBlob into which the URI is input.

Description

Outputs the contents of the URI. This method outputs the scheme and the following COLON, then the body of the URI. Angle brackets are *NOT* added by URIs.

3.4.1.25.3 - Enumerations

3.4.1.25.3.1 - IUri::ESpecialCharactersAllowed Enumeration

Enum to use in parameter to the IUri::Parse (see page 382) method.

C++

```
enum ESpecialCharactersAllowed {
    eDISALLOW_SPECIAL_CHARS,
    eALLOW_SPECIAL_CHARS
};
```

Description

Specifies if the URI can accept the characters '?', ',' or ';'. In most cases, this corresponds to the URI being within angle quotes or not.

Members

Members	Description
eDISALLOW_SPECIAL_CHARS	The URI CANNOT contain commas, semi-colons and question marks.
eALLOW_SPECIAL_CHARS	The URI can contain commas, semi-colons and question marks.

3.4.1.25.3.2 - IUri::EUriType Enumeration

All URI types must be listed in this enum.

C++

```
enum EUriType {
    eABSOLUTE,
    eSIP,
    eSIPS,
    eTEL,
    eMAILBOX,
    eIM,
    ePRES
};
```

Description

It is required because of the `IsEquivalent` (see page 382) method.

Members

Members	Description
eABSOLUTE	Any RFC2396/RFC3261-compliant URI that the stack does not support explicitly, for example http or mail.
eSIP	RFC3261 "sip" URI.
eSIPS	RFC3261 "sips" URI.
eTEL	RFC3966 "tel" URI.
eMAILBOX	RFC2822 mailbox URI.
eIM	RFC3860 "im" mailbox type URI.
ePRES	RFC3859 "pres" mailbox type URI.

3.4.2 - Enumerations

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

Enumerations

Enumeration	Description
ESipHeaderType (see page 384)	
ESipMethod (see page 387)	
ESipStatusClass (see page 388)	

3.4.2.1 - ESipHeaderType Enumeration

```
enum ESipHeaderType {
    eHDR_ACCEPT = 0,
    eHDR_ACCEPT_CONTACT,
    eHDR_ACCEPT_ENCODING,
    eHDR_ACCEPT_LANGUAGE,
    eHDR_ACCEPT_RESOURCE_PRIORITY,
    eHDR_ALERT_INFO,
    eHDR_ALLOW,
    eHDR_ALLOW_EVENTS,
    eHDR_AUTHENTICATION_INFO,
    eHDR_AUTHORIZATION,
    eHDR_CALL_ID,
    eHDR_CALL_INFO,
    eHDR_CONTACT,
    eHDR_CONTENT_DESCRIPTION,
    eHDR_CONTENT_DISPOSITION,
    eHDR_CONTENT_ENCODING,
```

```

eHDR_CONTENT_ID,
eHDR_CONTENT_LANGUAGE,
eHDR_CONTENT_LENGTH,
eHDR_CONTENT_TRANSFER_ENCODING,
eHDR_CONTENT_TYPE,
eHDR_CSEQ,
eHDR_DATE,
eHDR_DIVERSION,
eHDR_ERROR_INFO,
eHDR_EVENT,
eHDR_EXPIRES,
eHDR_FLOW_TIMER,
eHDR_FROM,
eHDR_HISTORY_INFO,
eHDR_IDENTITY,
eHDR_IDENTITY_INFO,
eHDR_IN_REPLY_TO,
eHDR_JOIN,
eHDR_MAX_FORWARDS,
eHDR_MIME_VERSION,
eHDR_MIN_EXPIRES,
eHDR_MIN_SE,
eHDR_ORGANIZATION,
eHDR_P_ACCESS_NETWORK_INFO,
eHDR_P_ASSERTED_IDENTITY,
eHDR_P_ASSOCIATED_URI,
eHDR_P_CALLED_PARTY_ID,
eHDR_P_CHARGING_FUNCTION_ADDRESSES,
eHDR_P_CHARGING_VECTOR,
eHDR_P_MEDIA_AUTHORIZATION,
eHDR_P_PREFERRED_IDENTITY,
eHDR_P_VISITED_NETWORK_ID,
eHDR_PATH,
eHDR_PRIORITY,
eHDR_PRIVACY,
eHDR_PROXY_AUTHENTICATE,
eHDR_PROXY_AUTHORIZATION,
eHDR_PROXY_REQUIRE,
eHDR_RACK,
eHDR_REASON,
eHDR_RECORD_ROUTE,
eHDR_REFERER_TO,
eHDRREFERRED_BY,
eHDR_REJECT_CONTACT,
eHDR_REPLACE,
eHDR_REPLY_TO,
eHDR_REQUEST_DISPOSITION,
eHDR_REQUIRE,
eHDR_RESOURCE_PRIORITY,
eHDR_RETRY_AFTER,
eHDR_ROUTE,
eHDR_RSEQ,
eHDR_SERVER,
eHDR_SERVICE_ROUTE,
eHDR_SESSION_EXPIRES,
eHDR_SIP_ETAG,
eHDR_SIP_IF_MATCH,
eHDR_SUBJECT,
eHDR_SUBSCRIPTION_STATE,
eHDR_SUPPORTED,
eHDR_TARGET_DIALOG,
eHDR_TIMESTAMP,
eHDR_TO,
eHDR_UNSUPPORTED,
eHDR_USER_AGENT,
eHDR_VIA,
eHDR_WARNING,
eHDR_WWW_AUTHENTICATE,
eHDR_EXTENSION
} ;

```

Description

The ESipHeaderType enum is the list of headers that the SIP stack understands. The eHDR_EXTENSION value is a special case and represents the generic form of header as per RFC 3261, and is used to extend the stack to work with headers that it does not know

about or cannot parse.

Warning

This enum is used to index a table of header descriptors.

Members

Members	Description
eHDR_ACCEPT = 0	RFC 3261 (SIP) headers.
eHDR_ACCEPT_CONTACT	RFC 3841 (Caller Preferences for the Session Initiation Protocol.)
eHDR_ACCEPT_ENCODING	RFC 3261 (SIP) headers.
eHDR_ACCEPT_LANGUAGE	RFC 3261 (SIP) headers.
eHDR_ACCEPT_RESOURCE_PRIORITY	RFC 4412 (Communications Resource Priority for the Session Initiation Protocol (SIP))
eHDR_ALERT_INFO	RFC 3261 (SIP) headers.
eHDR_ALLOW	RFC 3261 (SIP) headers.
eHDR_ALLOW_EVENTS	RFC 3265 (Event notification) headers.
eHDR_AUTHENTICATION_INFO	RFC 3261 (SIP) headers.
eHDR_AUTHORIZATION	RFC 3261 (SIP) headers.
eHDR_CALL_ID	RFC 3261 (SIP) headers.
eHDR_CALL_INFO	RFC 3261 (SIP) headers.
eHDR_CONTACT	RFC 3261 (SIP) headers.
eHDR_CONTENT_DESCRIPTION	RFC 2045 (Multipurpose Internet Mail Extensions (MIME) Part One) headers.
eHDR_CONTENT_DISPOSITION	RFC 3261 (SIP) headers.
eHDR_CONTENT_ENCODING	RFC 3261 (SIP) headers.
eHDR_CONTENT_ID	RFC 2111 (Content-ID and Message-ID Uniform Resource Locators) headers.
eHDR_CONTENT_LANGUAGE	RFC 3261 (SIP) headers.
eHDR_CONTENT_LENGTH	RFC 3261 (SIP) headers.
eHDR_CONTENT_TRANSFER_ENCODING	RFC 2045 (Multipurpose Internet Mail Extensions (MIME) Part One) headers.
eHDR_CONTENT_TYPE	RFC 3261 (SIP) headers.
eHDR_CSEQ	RFC 3261 (SIP) headers.
eHDR_DATE	RFC 3261 (SIP) headers.
eHDR_DIVERSION	Draft-levy-sip-diversion-08.txt.
eHDR_ERROR_INFO	RFC 3261 (SIP) headers.
eHDR_EVENT	RFC 3265 (Event notification) headers.
eHDR_EXPIRES	RFC 3261 (SIP) headers.
eHDR_FLOW_TIMER	draft-ietf-sip-outbound-15.txt
eHDR_FROM	RFC 3261 (SIP) headers.
eHDR_HISTORY_INFO	RFC 4244 (An Extension to the Session Initiation Protocol (SIP) for Request History Information)
eHDR_IDENTITY	draft-ietf-sip-identity-06.txt headers.
eHDR_IDENTITY_INFO	draft-ietf-sip-identity-06.txt headers.
eHDR_IN_REPLY_TO	RFC 3261 (SIP) headers.
eHDR_JOIN	RFC 3911 (The Session Initiation Protocol (SIP) "Join" Header)
eHDR_MAX_FORWARDS	RFC 3261 (SIP) headers.
eHDR_MIME_VERSION	RFC 3261 (SIP) headers.
eHDR_MIN_EXPIRES	RFC 3261 (SIP) headers.
eHDR_MIN_SE	RFC 4028 (Session Timers in the Session Initiation Protocol)
eHDR_ORGANIZATION	RFC 3261 (SIP) headers.
eHDR_P_ACCESS_NETWORK_INFO	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_P_ASSERTED_IDENTITY	RFC 3325 (Private Extensions to the SIP for Asserted Identity within Trusted Networks) headers.
eHDR_P_ASSOCIATED_URI	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_P_CALLED_PARTY_ID	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_P_CHARGING_FUNCTION_ADDRESSES	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_P_CHARGING_VECTOR	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_P_MEDIA_AUTHORIZATION	RFC 3313 (Private SIP Extensions for Media Authorization) headers.

eHDR_P_PREFERRED_IDENTITY	RFC 3325 (Private Extensions to the SIP for Asserted Identity within Trusted Networks) headers.
eHDR_P_VISITED_NETWORK_ID	RFC 3455 (P-Header Extensions to the SIP for the 3GPP) headers.
eHDR_PATH	RFC 3327 (SIP Extension Header Field for Registering Non-Adjacent Contacts) headers.
eHDR_PRIORITY	RFC 3261 (SIP) headers.
eHDR_PRIVACY	RFC 3323 (A Privacy Mechanism for the SIP) headers.
eHDR_PROXY_AUTHENTICATE	RFC 3261 (SIP) headers.
eHDR_PROXY_AUTHORIZATION	RFC 3261 (SIP) headers.
eHDR_PROXY_REQUIRE	RFC 3261 (SIP) headers.
eHDR_RACK	RFC 3262 (PRACK) headers.
eHDR_REASON	RFC 3326 (The Reason Header Field for the SIP) headers.
eHDR_RECORD_ROUTE	RFC 3261 (SIP) headers.
eHDR_REFER_TO	RFC 3515 (The SIP Refer Method) headers.
eHDRREFERRED_BY	RFC 3892 (Referred-By Mechanism)
eHDR_REJECT_CONTACT	RFC 3841 (Caller Preferences for the Session Initiation Protocol.)
eHDR_REPLACE	RFC 3891 (The SIP "Replaces" Header) headers.
eHDR_REPLY_TO	RFC 3261 (SIP) headers.
eHDR_REQUEST_DISPOSITION	RFC 3841 (Caller Preferences for the Session Initiation Protocol.)
eHDR_REQUIRE	RFC 3261 (SIP) headers.
eHDR_RESOURCE_PRIORITY	RFC 4412 (Communications Resource Priority for the Session Initiation Protocol (SIP))
eHDR_RETRY_AFTER	RFC 3261 (SIP) headers.
eHDR_ROUTE	RFC 3261 (SIP) headers.
eHDR_RSEQ	RFC 3262 (PRACK) headers.
eHDR_SERVER	RFC 3261 (SIP) headers.
eHDR_SERVICE_ROUTE	RFC 3608 (SIP Extension Header Field for Service Route Discovery During Registration) headers.
eHDR_SESSION_EXPIRES	RFC 4028 (Session Timers in the Session Initiation Protocol)
eHDR_SIP_ETAG	RFC 3903 (Session Initiation Protocol (SIP) Extension for Event State Publication)
eHDR_SIP_IF_MATCH	RFC 3903 (Session Initiation Protocol (SIP) Extension for Event State Publication)
eHDR_SUBJECT	RFC 3261 (SIP) headers.
eHDR_SUBSCRIPTION_STATE	RFC 3265 (Event notification) headers.
eHDR_SUPPORTED	RFC 3261 (SIP) headers.
eHDR_TARGET_DIALOG	RFC 4538 (Request Authorization through Dialog Identification in the Session Initiation Protocol (SIP)).
eHDR_TIMESTAMP	RFC 3261 (SIP) headers.
eHDR_TO	RFC 3261 (SIP) headers.
eHDR_UNSUPPORTED	RFC 3261 (SIP) headers.
eHDR_USER_AGENT	RFC 3261 (SIP) headers.
eHDR_VIA	RFC 3261 (SIP) headers.
eHDR_WARNING	RFC 3261 (SIP) headers.
eHDR_WWW_AUTHENTICATE	RFC 3261 (SIP) headers.
eHDR_EXTENSION	This is an application defined header (not related to any known RFC or draft). It is used when we don't recognize a header. This enum must be kept last. It maps to the RFC3261 extension-header construct.

3.4.2.2 - ESipMethod Enumeration

```
enum ESipMethod {
    eSIP_METHOD_ACK = 0,
    eSIP_METHOD_BYE,
    eSIP_METHOD_CANCEL,
    eSIP_METHOD_INFO,
    eSIP_METHOD_INVITE,
    eSIP_METHOD_MESSAGE,
    eSIP_METHOD_NOTIFY,
    eSIP_METHOD_OPTIONS,
    eSIP_METHOD_PING,
    eSIP_METHOD_PRACK,
    eSIP_METHOD_PUBLISH,
```

```

eSIP_METHOD_REFER,
eSIP_METHOD_REGISTER,
eSIP_METHOD_SERVICE,
eSIP_METHOD_SUBSCRIBE,
eSIP_METHOD_UPDATE,
eSIP_METHOD_UNKNOWN
};

```

Description

ESipMethod enumerates some of the various SIP methods that can be used with the stack.

Members

Members	Description
eSIP_METHOD_ACK = 0	RFC 3261 (SIP).
eSIP_METHOD_BYE	RFC 3261 (SIP).
eSIP_METHOD_CANCEL	RFC 3261 (SIP).
eSIP_METHOD_INFO	RFC 2976 (The SIP INFO Method).
eSIP_METHOD_INVITE	RFC 3261 (SIP).
eSIP_METHOD_MESSAGE	RFC 3428 (SIP Extension for Instant Messaging).
eSIP_METHOD_NOTIFY	RFC 3265 (SIP-Specific Event Notification).
eSIP_METHOD_OPTIONS	RFC 3261 (SIP).
eSIP_METHOD_PING	draft-sen-midcom-fw-nat-02.txt.
eSIP_METHOD_PRACK	RFC 3262 (PRACK).
eSIP_METHOD_PUBLISH	RFC 3903 (Session Initiation Protocol (SIP) Extension for Event State Publication)
eSIP_METHOD_REFER	RFC 3515 (The SIP Refer Method).
eSIP_METHOD_REGISTER	RFC 3261 (SIP).
eSIP_METHOD_SERVICE	draft-deason-sip-soap-00.txt.
eSIP_METHOD_SUBSCRIBE	RFC 3265 (SIP-Specific Event Notification).
eSIP_METHOD_UPDATE	RFC 3311 (SIP UPDATE Method).
eSIP_METHOD_UNKNOWN	A type of method that the SipStack is unaware of.

3.4.2.3 - ESipStatusClass Enumeration

```

enum ESipStatusClass {
    eSIP_STATUS_CLASS_INFORMATIONAL = 0,
    eSIP_STATUS_CLASS_SUCCESS,
    eSIP_STATUS_CLASS_REDIRECT,
    eSIP_STATUS_CLASS_CLIENT_ERROR,
    eSIP_STATUS_CLASS_SERVER_ERROR,
    eSIP_STATUS_CLASS_GLOBAL_FAILURE,
    eSIP_STATUS_CLASS_UNKNOWN
};

```

Description

This enumerate contains all RFC 3261 status classes.

Members

Members	Description
eSIP_STATUS_CLASS_INFORMATIONAL = 0	RFC 3261 Informational status class.
eSIP_STATUS_CLASS_SUCCESS	RFC 3261 Success status class.
eSIP_STATUS_CLASS_REDIRECT	RFC 3261 Redirection status class.
eSIP_STATUS_CLASS_CLIENT_ERROR	RFC 3261 Client-Error status class.
eSIP_STATUS_CLASS_SERVER_ERROR	RFC 3261 Server-Error status class.
eSIP_STATUS_CLASS_GLOBAL_FAILURE	RFC 3261 Global-Failure status class.
eSIP_STATUS_CLASS_UNKNOWN	Non-RFC 3261 status class.

3.4.3 - Functions

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

Functions

Function	Description
MxConvertSipHeader (see page 389)	This method converts a header name to its enum form.
MxConvertSipMethod (see page 389)	Converts a method string to the equivalent enum.
MxConvertSipMethod (see page 390)	Converts a method enum to its equivalent string.
MxGetDefaultReasonPhrase (see page 390)	Finds the default reason phrase associated with the status code.
MxGetSipStatusClass (see page 391)	Finds in which status class interval a specific status code can be found.
MxSetDefaultHeaderOrder (see page 391)	Resets the header order to its default.

3.4.3.1 - MxConvertSipHeader Function

This method converts a header name to its enum form.

C++

```
ESipHeaderType MxConvertSipHeader( IN const CString& rstrHeader );
```

Parameters

Parameters	Description
IN const CString& rstrHeader	Name to convert.

Returns

Header type, or eHDR_EXTENSION if not found.

Description

This method converts the name string to its equivalent in the ESipHeaderType (see page 384) enum. If the given name is one character form, short form is assumed, and this method then checks only those header types that have a short form.

Header name comparison is case-insensitive.

```
RFC 3261 ABNF:
extension-header = header-name HCOLON header-value
header-name     = token
```

```
RFC 3261 Section 7.3.1 pg. 32:
When comparing header fields, field names are always case-insensitive.
```

If the name is not found in the table of known headers, eHDR_EXTENSION is returned.

See Also

Header definition table

3.4.3.2 - MxConvertSipMethod Function

Converts a method string to the equivalent enum.

C++

```
ESipMethod MxConvertSipMethod( IN const CString& rstrMethod );
```

Parameters

Parameters	Description
IN const CString& rstrMethod	Method to convert, in string form.

Returns

Enum version of the method string. eSIP_METHOD_UNKNOWN if the method is unknown or invalid.

Description

Finds the enum form of the method string. Method names are case sensitive.

```
RFC 3261 ABNF:
INVITEm      = %x49.4E.56.49.54.45 ; INVITE in caps
ACKm        = %x41.43.4B ; ACK in caps
```

```

OPTIONSm      = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm         = %x42.59.45 ; BYE in caps
CANCELm      = %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm    = %x52.45.47.49.53.54.45.52 ; REGISTER in caps
Method        = INVITEm / ACKm / OPTIONSm / BYEm /
                CANCELm / REGISTERm /
                extension-method
extension-method = token

RFC 3262 ABNF:
PRACKm      = %x50.52.41.43.4B ; PRACK in caps
Method        = INVITEm / ACKm / OPTIONSm / BYEm /
                CANCELm / REGISTERm / PRACKm
                / extension-method

RFC 3265 ABNF:
SUBSCRIBEm   = %x53.55.42.53.43.52.49.42.45 ; SUBSCRIBE in caps
NOTIFYm      = %x4E.4F.54.49.46.59 ; NOTIFY in caps
extension-method = SUBSCRIBEm / NOTIFYm / token

RFC 3311 ABNF:
UPDATEm     = %x55.50.44.41.54.45 ; UPDATE in caps
Method        = INVITEm / ACKm / OPTIONSm / BYEm /
                CANCELm / REGISTERm / UPDATEm
                / extension-method

RFC 3428 ABNF:
MESSAGEm    = %x4D.45.53.53.41.47.45 ;MESSAGE in caps

draft-ietf-sip-publish-04.txt ABNF:
PUBLISHm    = %x50.55.42.4C.49.53.48 ; PUBLISH in caps.
extension-method = PUBLISHm / token

```

See Also

Table `aszMETHOD_NAME`.

3.4.3.3 - `MxConvertSipMethod` Function

Converts a method enum to its equivalent string.

C++

```
const char* MxConvertSipMethod(IN ESipMethod eMethod);
```

Parameters

Parameters	Description
IN ESipMethod eMethod	Method to convert.

Returns

String version of the method. Empty string if `eMethod` is `eSIP_METHOD_UNKNOWN`.

Description

Returns the string form of a method name.

See Also

Table `g_aszMETHOD_NAME` (see page 391).

3.4.3.4 - `MxGetDefaultReasonPhrase` Function

Finds the default reason phrase associated with the status code.

C++

```
const char* MxGetDefaultReasonPhrase(IN uint16_t uCode);
```

Parameters

Parameters	Description
IN uint16_t uCode	Status code to convert.

Returns

Default reason string associated with the status code as per RFC 3261 and appropriate RFCs/drafts. Empty string for unknown codes.

Description

Finds the default reason phrase associated with the status code.

See Also

[MxGetSipStatusClass](#) (see page 391)

3.4.3.5 - MxGetSipStatusClass Function

Finds in which status class interval a specific status code can be found.

C++

```
ESipStatusClass MxGetSipStatusClass(IN uint16_t uCode);
```

Parameters

Parameters	Description
IN uint16_t uCode	Status code to convert.

Returns

Status class of the status code. eSIP_STATUS_CLASS_UNKNOWN if the status code that does not fit into any known class.

Description

This method finds in which status class interval a specific status code can be found.

```
RFC 3261 ABNF:
Status-Code      =  Informational
                  /  Redirection
                  /  Success
                  /  Client-Error
                  /  Server-Error
                  /  Global-Failure
                  /  extension-code
```

3.4.3.6 - MxSetDefaultHeaderOrder Function

Resets the header order to its default.

C++

```
void MxSetDefaultHeaderOrder();
```

Description

This method initializes the header order table. The default header order number for each header is its enum value times 1000. Doing so gives a lot of room for users to reorder the headers as they see fit. The headers used by proxies are ordered at the start of the list.

This method can be called by the application to reset the order to its default value after it changed it. It is automatically called by the CSipStackInitializer::Initialize (see page 660)() method.

See Also

[Table g_auHeaderOrder](#) (see page 392).

3.4.4 - Variables

This section documents the variables of the Sources/SipParser folder.

3.4.4.1 - g_aszMETHOD_NAME Variable

```
const char* const g_aszMETHOD_NAME[eSIP_METHOD_UNKNOWN+1];
```

Description

This array holds the actual strings representing some of the various SIP methods supported by the stack. Note that this array is indexed

according to the ESipMethod (see page 387) enumeration. Thus, g_aszMETHOD_NAME[eREGISTER] yields the "REGISTER" string.

3.4.4.2 - g_auHeaderOrder Variable

```
uint32_t g_auHeaderOrder[];
```

Description

This table contains the order of headers. All headers must have a different order number. By default, the header order is that of the ESipHeaderType (see page 384) enum, except for the header types that proxies use.

The default header order number for each header is its (ESipHeaderType (see page 384)) enum value times 1000. This means that there is a gap of 999 ordering slots between each header type. Doing so gives a lot of room for users to reorder the headers as they see fit. To reorder headers, users can simply use the g_auHeaderOrder table directly and set a value for the wanted ESipHeaderType (see page 384) type. Care must be taken that all header types must have different order values, otherwise the sorting algorithm may produce unexpected results.

RFC 3261 Section 7.3.1

The relative order of header fields with different field names is not significant. However, it is RECOMMENDED that header fields which are needed **for** proxy processing (Via, Route, Record-Route, Proxy-Require, Max-Forwards, and Proxy-Authorization, **for** example) appear towards the top of the message to facilitate rapid parsing. The relative order of header field rows with the same field name is important.

Warning

This table is indexed by the ESipHeaderType (see page 384) enum.

Also see: MxSetDefaultHeaderOrder (see page 391)

3.4.4.3 - Status Codes

This file contains the SIP status codes and reason phrases associated with them. It also contains enums and helper methods.

Location

SipParser/StatusCode.h

3.4.4.3.1 - RFC 3265 (Event notification) status codes

```
const uint8_t uACCEPTABLE = 202;
const char* const szACCEPTABLE = "Accepted";
const uint16_t uBAD_EVENT = 489;
const char* const szBAD_EVENT = "Bad Event";
```

Description

Status codes defined in RFC 3265.

3.4.4.3.2 - draft-ietf-sip-publish-04.txt status codes

```
const uint16_t uCONDITIONAL_REQUEST_FAILED = 412;
const char* const szCONDITIONAL_REQUEST_FAILED = "Conditional Request Failed";
```

Description

Status codes defined in draft-ietf-sip-publish-04.txt.

3.4.4.3.3 - draft-ietf-sip-referredby-05.txt status codes

```
const uint16_t uPROVIDE_REFERRER_IDENTITY = 429;
const char* const szPROVIDE_REFERRER_IDENTITY = "Provide Referrer Identity";
```

Description

Status codes defined in draft-ietf-sip-referredby-05.txt.

3.4.4.3.4 - draft-ietf-sip-session-timer-14.txt status codes

```
const uint16_t uSESSION_INTERVAL_TOO_SMALL = 422;
const char* const szSESSION_INTERVAL_TOO_SMALL = "Session Interval Too Small";
```

Description

Status codes defined in draft-ietf-sip-session-timer-14.txt.

3.4.4.3.5 - RFC 3261 Status codes

```
const uint8_t uTRYING = 100;
const char* const szTRYING = "Trying";
const uint8_t uRINGING = 180;
const char* const szRINGING = "Ringing";
const uint8_t uCALL_IS_BEING_FORWARDED = 181;
const char* const szCALL_IS_BEING_FORWARDED = "Call Is Being Forwarded";
const uint8_t uQUEUED = 182;
const char* const szQUEUED = "Queued";
const uint8_t uSESSION_PROGRESS = 183;
const char* const szSESSION_PROGRESS = "Session Progress";
const uint8_t uOK = 200;
const char* const szOK = "OK";
const uint16_t uMULTIPLE_CHOICES = 300;
const char* const szMULTIPLE_CHOICES = "Multiple Choices";
const uint16_t uMOVED_PERMANENTLY = 301;
const char* const szMOVED_PERMANENTLY = "Moved Permanently";
const uint16_t uMOVED_TEMPORARILY = 302;
const char* const szMOVED_TEMPORARILY = "Moved Temporarily";
const uint16_t uUSE_PROXY = 305;
const char* const szUSE_PROXY = "Use Proxy";
const uint16_t uALTERNATIVE_SERVICE = 380;
const char* const szALTERNATIVE_SERVICE = "Alternative Service";
const uint16_t uBAD_REQUEST = 400;
const char* const szBAD_REQUEST = "Bad Request";
const char* const szBAD_REQUEST_INVALID_CONTACT = "Missing, Erroneous or Multiple Contact header field(s)";
const char* const szBAD_REQUEST_INVALID_MESSAGE_SUMMARY = "Missing or invalid message-summary";
const char* const szINVALID_SUBSCRIPTION_STATE = "Invalid Subscription-State Header";
const char* const szMISSING_SUBSCRIPTION_STATE = "Missing Subscription-State Header";
const char* const szINVALID_P_ASSERTED_IDENTITY = "Too many or erroneous P-Asserted-Identity header(s)";
const uint16_t uUNAUTHORIZED = 401;
const char* const szUNAUTHORIZED = "Unauthorized";
const uint16_t uPAYMENT_REQUIRED = 402;
const char* const szPAYMENT_REQUIRED = "Payment Required";
const uint16_t uFORBIDDEN = 403;
const char* const szFORBIDDEN = "Forbidden";
const uint16_t uNOT_FOUND = 404;
const char* const szNOT_FOUND = "Not Found";
const uint16_t uMETHOD_NOT_ALLOWED = 405;
const char* const szMETHOD_NOT_ALLOWED = "Method Not Allowed";
const uint16_t uNOT_ACCEPTABLE = 406;
const char* const szNOT_ACCEPTABLE = "Not Acceptable";
const uint16_t uPROXY_AUTHENTICATION_REQUIRED = 407;
const char* const szPROXY_AUTHENTICATION_REQUIRED = "Proxy Authentication Required";
const uint16_t uREQUEST_TIMEOUT = 408;
const char* const szREQUEST_TIMEOUT = "Request Timeout";
const uint16_t uGONE = 410;
const char* const szGONE = "Gone";
const uint16_t uREQUEST_ENTITY_TOO_LARGE = 413;
const char* const szREQUEST_ENTITY_TOO_LARGE = "Request Entity Too Large";
const uint16_t uREQUEST_URI_TOO_LARGE = 414;
const char* const szREQUEST_URI_TOO_LARGE = "Request-URI Too Large";
const uint16_t uUNSUPPORTED_MEDIA_TYPE = 415;
const char* const szUNSUPPORTED_MEDIA_TYPE = "Unsupported Media Type";
const uint16_t uUNSUPPORTED_URI_SCHEME = 416;
const char* const szUNSUPPORTED_URI_SCHEME = "Unsupported URI Scheme";
const uint16_t uBAD_EXTENSION = 420;
const char* const szBAD_EXTENSION = "Bad Extension";
const uint16_t uEXTENSION_REQUIRED = 421;
const char* const szEXTENSION_REQUIRED = "Extension Required";
const uint16_t uINTERVAL_TOO_BRIEF = 423;
const char* const szINTERVAL_TOO_BRIEF = "Interval Too Brief";
const uint16_t uTEMPORARILY_NOT_AVAILABLE = 480;
```

```

const char* const szTEMPORARILY_NOT_AVAILABLE = "Temporarily not available";
const uint16_t uCALL_LEG_TRANSACTION_DOES_NOT_EXIST = 481;
const char* const szCALL_LEG_TRANSACTION_DOES_NOT_EXIST = "Call Leg/Transaction Does Not Exist";
const uint16_t uLOOP_DETECTED = 482;
const char* const szLOOP_DETECTED = "Loop Detected";
const uint16_t uTOO_MANY_HOPS = 483;
const char* const szTOO_MANY_HOPS = "Too Many Hops";
const uint16_t uADDRESS_INCOMPLETE = 484;
const char* const szADDRESS_INCOMPLETE = "Address Incomplete";
const uint16_t uAMBIGUOUS = 485;
const char* const szAMBIGUOUS = "Ambiguous";
const uint16_t uBUSY_HERE = 486;
const char* const szBUSY_HERE = "Busy Here";
const uint16_t uREQUEST_TERMINATED = 487;
const char* const szREQUEST_TERMINATED = "Request Terminated";
const uint16_t uNOT_ACCEPTABLE_HERE = 488;
const char* const szNOT_ACCEPTABLE_HERE = "Not Acceptable Here";
const char* const szUNSUPPORTED_REFER_TO_URI_SCHEME = "Unsupported Refer-To URI Scheme";
const uint16_t uREQUEST_PENDING = 491;
const char* const szREQUEST_PENDING = "Request Pending";
const uint16_t uUNDECIPHERABLE = 493;
const char* const szUNDECIPHERABLE = "Undecipherable";
const uint16_t uINTERNAL_SERVER_ERROR = 500;
const char* const szINTERNAL_SERVER_ERROR = "Internal Server Error";
const uint16_t uNOT_IMPLEMENTED = 501;
const char* const szNOT_IMPLEMENTED = "Not Implemented";
const uint16_t uBAD_GATEWAY = 502;
const char* const szBAD_GATEWAY = "Bad Gateway";
const uint16_t uSERVICE_UNAVAILABLE = 503;
const char* const szSERVICE_UNAVAILABLE = "Service Unavailable";
const uint16_t uSERVER_TIME_OUT = 504;
const char* const szSERVER_TIME_OUT = "Server Time-out";
const uint16_t uSIP_VERSION_NOT_SUPPORTED = 505;
const char* const szSIP_VERSION_NOT_SUPPORTED = "SIP Version not supported";
const uint16_t uMESSAGE_TOO_LARGE = 513;
const char* const szMESSAGE_TOO_LARGE = "Message Too Large";
const uint16_t uBUSY_EVERYWHERE = 600;
const char* const szBUSY_EVERYWHERE = "Busy Everywhere";
const uint16_t uDECLINE = 603;
const char* const szDECLINE = "Decline";
const uint16_t uDOES_NOT_EXIST_ANYWHERE = 604;
const char* const szDOES_NOT_EXIST_ANYWHERE = "Does not exist anywhere";
const uint16_t uGLOBAL_NOT_ACCEPTABLE = 606;
const char* const szGLOBAL_NOT_ACCEPTABLE = szNOT_ACCEPTABLE;

```

Description

Status codes defined in RFC 3261.

3.4.4.3.6 - draft-ietf-sip-resource-priority-03.txt status codes

```

const uint16_t uUNKNOWN_RESOURCE_PRIORITY = 417;
const char* const szUNKNOWN_RESOURCE_PRIORITY = "Unknown Resource-Priority";

```

Description

Status codes defined in draft-ietf-sip-resource-priority-03.txt.

3.4.4.4 - Uri Parameter

This file contains the SIP URI parameter names and some pre-defined values.

Location

SipParser/UriParameter.h

3.4.4.4.1 - RFC 3486 URI parameter names

```

const char* const szURIPARAM_COMP = "comp";
const char* const szURIPARAM_COMP_VALUE_SIGCOMP = "sigcomp";

```

Description

Additional URI parameter names defined in RFC 3486 but not defined in RFC 3261.

3.4.4.4.2 - RFC 3261 URI parameter names

```
const char* const szURIPARAM_LR = "lr";
const char* const szURIPARAM_MADDR = "maddr";
const char* const szURIPARAM_METHOD = "method";
const char* const szURIPARAM_TRANSPORT = "transport";
const char* const szURIPARAM_TTL = "ttl";
const char* const szURIPARAM_USER = "user";
```

Description

URI parameter names defined in RFC 3261.

3.4.4.4.3 - RFC 3966 URI parameter names

```
const char* const szURIPARAM_PHONE_CONTEXT = "phone-context";
```

Description

URI parameter names defined in RFC 3966.

3.4.4.4.4 - RFC3261 "transport" URI parameter values

```
const char* const szURIPARAM_TRANSPORT_UDP = "udp";
const char* const szURIPARAM_TRANSPORT_TCP = "tcp";
const char* const szURIPARAM_TRANSPORT_TLS = "tls";
```

Description

Values defined in RFC 3261 for the "transport" URI parameter.

3.4.4.4.5 - RFC3261 "user" URI parameter values

```
const char* const szURIPARAM_USER_VALUE_IP = "ip";
const char* const szURIPARAM_USER_VALUE_PHONE = "phone";
```

Description

Values defined in RFC 3261 for the "user" URI parameter.

3.4.4.5 - Header Parameters

This file contains the SIP header parameter names and some pre-defined values.

Location

SipParser/HeaderParameter.h

3.4.4.5.1 - RFC 1847 parameter names

```
const char* const pszHDRPARAM_BOUNDARY = "boundary";
```

Description

Parameter names defined in RFC 1847 but not defined in RFC 3261.

3.4.4.5.2 - Draft-levy-sip-diversion-08 Header parameter names

```
const char* const pszHDRPARAM_COUNTER = "counter";
const char* const pszHDRPARAM_LIMIT = "limit";
const char* const pszHDRPARAM_PRIVACY = "privacy";
const char* const pszHDRPARAM_SCREEN = "screen";
```

Description

Additional header parameter names defined in Draft-levy-sip-diversion-08 but not defined in RFC3261.

3.4.4.5.3 - Draft-ietf-sip-outbound-15 Header parameter names

```
const char* const pszHDRPARAM_REG_ID = "reg-id";
const char* const pszHDRPARAM_SIP_INSTANCE = "+sip.instance";
const char* const pszHDRPARAM_OB = "ob";
```

Description

Additional header parameter names defined in Draft-ietf-sip-outbound-15 but not defined in RFC3261.

3.4.4.5.4 - RFC 4538 Header parameter names

```
const char* const pszHDRPARAM_REMOTE_TAG = "remote-tag";
const char* const pszHDRPARAM_LOCAL_TAG = "local-tag";
```

Description

Additional header parameter names defined in RFC 4538 but not defined in RFC 3261.

3.4.4.5.5 - RFC 3329 Header parameter names

```
const char* const szHDRPARAM_ALG = "alg";
const char* const szHDRPARAM_D_ALG = "d-alg";
const char* const szHDRPARAM_D_QOP = "d-qop";
const char* const szHDRPARAM_D_VER = "d-ver";
const char* const szHDRPARAM_EALG = "ealg";
const char* const szHDRPARAM_MOD = "mod";
const char* const szHDRPARAM_PORT1 = "port1";
const char* const szHDRPARAM_PORT2 = "port2";
const char* const szHDRPARAM_PROT = "prot";
const char* const szHDRPARAM_SPI = "api";
```

Description

Additional header parameter names defined in RFC 3329 but not defined in RFC 3261.

3.4.4.5.6 - RFC 3261 Header parameter names

```
const char* const szHDRPARAM_ALGORITHM = "algorithm";
const char* const szHDRPARAM_BRANCH = "branch";
const char* const szHDRPARAM_CNONCE = "cnonce";
const char* const szHDRPARAM_DOMAIN = "domain";
const char* const szHDRPARAM_DURATION = "duration";
const char* const szHDRPARAM_EXPIRES = "expires";
const char* const szHDRPARAM_HANDLING = "handling";
const char* const szHDRPARAM_MADDR = "maddr";
const char* const szHDRPARAM_NC = "nc";
const char* const szHDRPARAM_NEXTNONCE = "nextnonce";
const char* const szHDRPARAM_NONCE = "nonce";
const char* const szHDRPARAM_OPAQUE = "opaque";
const char* const szHDRPARAM_PURPOSE = "purpose";
const char* const szHDRPARAM_Q = "q";
const char* const szHDRPARAM_QOP = "qop";
const char* const szHDRPARAM_REALM = "realm";
const char* const szHDRPARAM_RECEIVED = "received";
const char* const szHDRPARAM_RESPONSE = "response";
const char* const szHDRPARAM_RSPAUTH = "rspauth";
const char* const szHDRPARAM_STALE = "stale";
const char* const szHDRPARAM_TAG = "tag";
const char* const szHDRPARAM_TTL = "ttl";
const char* const szHDRPARAM_URI = "uri";
const char* const szHDRPARAM_USERNAME = "username";
```

Description

Header parameter names defined in RFC 3261.

3.4.4.5.7 - RFC3261 "algorithm" header parameter values

```
const char* const szHDRPARAM_ALGORITHM_VALUE_MD5 = "MD5";
const char* const szHDRPARAM_ALGORITHM_VALUE_MD5_SESS = "MD5-sess";
```

Description

Values defined in RFC 3261 for the "algorithm" header parameter.

3.4.4.5.8 - RFC 3310 Header parameter names

```
const char* const szHDRPARAM_AUTS = "auts";
```

Description

Additional header parameter names defined in RFC 3310 but not defined in RFC 3261.

3.4.4.5.9 - RFC3261 Magic cookie in "branch" parameter

```
const char* const szHDRPARAM_BRANCH_VALUE_MAGIC_COOKIE = "z9hG4bK";
```

Description

"branch" parameter value for a Via header. Defined in RFC 3261. If the "branch" parameter value of a Via header starts with this string, it means that the branch ID was built following RFC 3261.

3.4.4.5.10 - RFC 3603 Header parameter names

```
const char* const szHDRPARAM_CALLED = "called";
const char* const szHDRPARAM_CALLING = "calling";
const char* const szHDRPARAM_CHARGE = "charge";
const char* const szHDRPARAM_CONTENT = "content";
const char* const szHDRPARAM_COUNT = "count";
const char* const szHDRPARAM_KEY = "key";
const char* const szHDRPARAM_LOCROUTE = "locroute";
const char* const szHDRPARAM_REDIRECTOR_URI = "redirector-uri";
const char* const szHDRPARAM_RKSGROUP = "rksgroup";
const char* const szHDRPARAM_ROUTING = "routing";
```

Description

Additional header parameter names defined in RFC 3603 but not defined in RFC 3261.

3.4.4.5.11 - RFC 3326 Header parameter names

```
const char* const szHDRPARAM_CAUSE = "cause";
const char* const szHDRPARAM_TEXT = "test";
```

Description

Additional header parameter names defined in RFC 3326 but not defined in RFC 3261.

3.4.4.5.12 - RFC 3455 Header parameter names

```
const char* const szHDRPARAM_CCF = "ccf";
const char* const szHDRPARAM_CGI_3GPP = "cgi-3gpp";
const char* const szHDRPARAM_ECF = "ecf";
const char* const szHDRPARAM_ICID_VALUE = "icid-value";
const char* const szHDRPARAM_ICID_GENERATED_AT = "icid-generated-at";
const char* const szHDRPARAM_ORIG_IOI = "orig-ioi";
const char* const szHDRPARAM_TERM_IOI = "term-ioi";
const char* const szHDRPARAM_UTRAN_CELL_ID_3GPP = "utran-cell-id-3gpp";
```

Description

Additional header parameter names defined in RFC 3455 but not defined in RFC 3261.

3.4.4.5.13 - RFC 3486 Header parameter names

```
const char* const szHDRPARAM_COMP = "comp";
```

Description

Additional header parameter names defined in RFC 3486 but not defined in RFC 3261.

3.4.4.5.14 - RFC 3891 Header parameter names

```
const char* const szHDRPARAM_EARLY_ONLY = "early-only";
const char* const szHDRPARAM_FROM_TAG = "from-tag";
const char* const szHDRPARAM_TO_TAG = "to-tag";
```

Description

Additional header parameter names defined in RFC 3891 but not defined in RFC 3261.

3.4.4.5.15 - RFC 3265 Header parameter names

```
const char* const szHDRPARAM_ID = "id";
const char* const szHDRPARAM_REASON = "reason";
const char* const szHDRPARAM_RETRY_AFTER = "retry-after";
```

Description

Additional header parameter names defined in RFC 3265 but not defined in RFC 3261.

3.4.4.5.16 - RFC3261 "qop" header parameter values

```
const char* const szHDRPARAM_QOP_VALUE_AUTH = "auth";
const char* const szHDRPARAM_QOP_VALUE_AUTH_INT = "auth-int";
```

Description

Values defined in RFC 3261 for the "qop" header parameter.

3.4.4.5.17 - draft-ietf-sip-session-timer-15 Header parameter names

```
const char* const szHDRPARAM_REFRESHER = "refresher";
const char* const szHDRPARAM_REFRESHER_VALUE_UAC = "uac";
const char* const szHDRPARAM_REFRESHER_VALUE_UAS = "uas";
```

Description

Additional header parameter names defined in draft-ietf-sip-session-timer-15 but not defined in RFC 3261.

3.4.4.5.18 - RFC 3581 Header parameter names

```
const char* const szHDRPARAM_RPORT = "rport";
```

Description

Additional header parameter names defined in RFC 3581 but not defined in RFC 3261.

3.4.4.6 - IUri Constants

This file contains some constants that are used by IUri (see page 380).

Location

SipParser/IUri.h

See Also

IUri (see page 380)

3.4.4.6.1 - szURISCHEME_IM Variable

```
const char* const szURISCHEME_IM = "im";
```

Description

RFC 3860 "im" URI scheme name. See IUri::EUriType::eIM (see page 384).

3.4.4.6.2 - szURISCHEME_PRES Variable

```
const char* const szURISCHEME_PRES = "pres";
```

Description

RFC 3859 "pres" URI scheme name. See IUri::EUriType::ePRES (see page 384).

3.4.4.6.3 - szURISCHEME_SIP Variable

```
const char* const szURISCHEME_SIP = "sip";
```

Description

RFC3261 "sip" URI scheme name. See IUri::EUriType::eSIP (see page 384).

3.4.4.6.4 - szURISCHEME_SIPS Variable

```
const char* const szURISCHEME_SIPS = "sips";
```

Description

RFC3261 "sips" URI scheme name. See IUri::EUriType::eSIPS (see page 384).

3.4.4.6.5 - szURISCHEME_TEL Variable

```
const char* const szURISCHEME_TEL = "tel";
```

Description

RFC3966 "tel" URI scheme name. See IUri::EUriType::eTEL (see page 384).

3.4.4.7 - RegInfo Constants

This file contains some constants that are used by CRegInfo.

Location

SipParser/CRegInfo.h

See Also

CReginfo (see page 222)

3.4.4.7.1 - Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo namespaces

```
const char* const pszNAMESPACE_GRUU = "urn:ietf:params:xml:ns:gruuinfo";
```

Description

The following namespaces are those supported by RFC 3680 and draft-ietf-sipping-gruu-reg-event-08.

3.4.4.7.2 - Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo attribute names

```
const char* const pszREGINFO_ATTRIBUTE_VERSION = "version";
const char* const pszREGINFO_ATTRIBUTE_STATE = "state";
const char* const pszREGINFO_ATTRIBUTE_XMLNS = "xmlns";
const char* const pszREGINFO_ATTRIBUTE_XMLNSGR = "xmlns:gr";
```

```
const char* const pszREGINFO_ATTRIBUTE_AOR = "aor";
const char* const pszREGINFO_ATTRIBUTE_ID = "id";
const char* const pszREGINFO_ATTRIBUTE_EVENT = "event";
const char* const pszREGINFO_ATTRIBUTE_DURATIONREGISTERED = "duration-registered";
const char* const pszREGINFO_ATTRIBUTE_RELATIVEPRIORITY = "q";
const char* const pszREGINFO_ATTRIBUTE_EXPIRES = "expires";
const char* const pszREGINFO_ATTRIBUTE_RETRY_AFTER = "retry-after";
const char* const pszREGINFO_ATTRIBUTE_CALLID = "callid";
const char* const pszREGINFO_ATTRIBUTE_CSEQ = "cseq";
const char* const pszREGINFO_ATTRIBUTE_DISPLAYNAME = "display-name";
const char* const pszREGINFO_ATTRIBUTE_NAME = "name";
```

Description

The following attribute names are those supported by RFC 3680 and draft-ietf-sipping-gruu-reg-event-08.

3.4.4.7.3 - Reginfo: RFC 3680 contact event values

```
const char* const pszREGINFO_CONTACT_EVENT_REGISTERED = "registered";
const char* const pszREGINFO_CONTACT_EVENT_CREATED = "created";
const char* const pszREGINFO_CONTACT_EVENT_REFRESHED = "refreshed";
const char* const pszREGINFO_CONTACT_EVENT_SHORTENED = "shortened";
const char* const pszREGINFO_CONTACT_EVENT_EXPIRED = "expired";
const char* const pszREGINFO_CONTACT_EVENT_DEACTIVATED = "deactivated";
const char* const pszREGINFO_CONTACT_EVENT_PROBATION = "probation";
const char* const pszREGINFO_CONTACT_EVENT_UNREGISTERED = "unregistered";
const char* const pszREGINFO_CONTACT_EVENT_REJECTED = "rejected";
```

Description

According to RFC 3680, the following events are those that can make a contact change state.

3.4.4.7.4 - RFC 3680 contact state values

```
const char* const pszREGINFO_CONTACT_STATE_ACTIVE = "active";
const char* const pszREGINFO_CONTACT_STATE_TERMINATED = "terminated";
```

Description

According to RFC 3680, there are two valid values for the state of the contact: active and terminated.

3.4.4.7.5 - Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo element names

```
const char* const pszREGINFO_ELEMENT_REGINFO = "reginfo";
const char* const pszREGINFO_ELEMENT_REGISTRATION = "registration";
const char* const pszREGINFO_ELEMENT_CONTACT = "contact";
const char* const pszREGINFO_ELEMENT_URI = "uri";
const char* const pszREGINFO_ELEMENT_UNKNOWNPARAM = "unknown-param";
const char* const pszREGINFO_ELEMENT_PUBGRUU = "pub-gruu";
const char* const pszREGINFO_ELEMENT_TEMPGRUU = "temp-gruu";
```

Description

The following element names are those supported by RFC 3680 and draft-ietf-sipping-gruu-reg-event-08.

3.4.4.7.6 - RFC 3680 registration state values

```
const char* const pszREGINFO_REGISTRATION_STATE_INIT = "init";
const char* const pszREGINFO_REGISTRATION_STATE_ACTIVE = "active";
const char* const pszREGINFO_REGISTRATION_STATE_TERMINATED = "terminated";
```

Description

According to RFC 3680, the state of the registration must be one of the following: init, active or terminated.

3.4.4.7.7 - RFC 3680 reginfo state values

```
const char* const pszREGINFO_STATE_PARTIAL = "partial";
const char* const pszREGINFO_STATE_FULL = "full";
```

Description

According to RFC 3680, there are two valid values for the state of the reginfo: full and partial

3.4.4.7.8 - Reginfo: Invalid values for contact's attributes

```
const unsigned int uREGINFO_CONTACT_INVALID_DURATION_REGISTERED = ~0U;
const unsigned int uREGINFO_CONTACT_INVALID_EXPIRES = ~0U;
const unsigned int uREGINFO_CONTACT_INVALID_RETRY_AFTER = ~0U;
const unsigned int uREGINFO_CONTACT_INVALID_CSEQ = ~0U;
```

Description

Invalid values for contact's attributes.

3.4.4.8 - CSipHeader Constants

This file contains some constants that are used by CSipHeader (see page 239).

Location

SipParser/CSipHeader.h

See Also

CSipHeader (see page 239)

3.4.4.8.1 - pszX_M5T_ORIGIN_HEADER_NAME Variable

```
const char* const pszX_M5T_ORIGIN_HEADER_NAME = "X-M5T-Origin";
```

Description

Define the name of the differentiator header.

See Also

MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR (see page 15)

3.4.4.8.2 - pszX_M5T_ORIGIN_HEADER_VALUE Variable

```
const char* const pszX_M5T_ORIGIN_HEADER_VALUE = "locally generated";
```

Description

Define the value for the differentiator header.

See Also

MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR (see page 15)

3.4.4.8.3 - uHEADERS_PARSE_ALL Variable

```
const uint8_t uHEADERS_PARSE_ALL = 0;
```

Description

Specifying this value for the uMaxHeaders parameter of Parse() will parse all available headers.

3.5 - SipProxy

This section documents the Sources/SipProxy folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 402)

3.5.1 - Classes

This section documents the classes of the Sources/SipProxy folder.

Classes

Class	Description
CSipProxyConfig (see page 402)	
ISipSessionStatefulProxyMgr (see page 405)	
ISipSessionStatefulProxySvc (see page 414)	
ISipStatelessProxyMgr (see page 420)	
ISipStatelessProxySvc (see page 425)	
ISipTransactionStatefulProxyMgr (see page 428)	
ISipTransactionStatefulProxySvc (see page 435)	

3.5.1.1 - CSipProxyConfig Class

Class Hierarchy



C++

```
class CSipProxyConfig;
```

Description

This class defines the necessary configuration for the application in order to implement a SIP Proxy. The application gives an instance of this class to the proxy services when creating one.

This class allows to set the proxy ID and the host addresses for which the Proxy is responsible.

Location

SipProxy/CSipProxyConfig.h

See Also

ISipStatelessProxySvc (see page 425), ISipTransactionStatefulProxySvc (see page 435)

Constructors

Constructor	Description
CSipProxyConfig (see page 402)	Default Constructor.

Legend



Methods

Method	Description
AddRef (see page 403)	Adds a reference to the object.
GetAddresses (see page 403)	Gets the addresses for which the Proxy is responsible.
GetAuthenticationRealm (see page 404)	Gets the Realm that the proxy manages.
GetProxyId (see page 404)	Gets the Proxy ID.
Release (see page 405)	Releases a reference to the object.

Legend



3.5.1.1.1 - Constructors

3.5.1.1.1.1 - CSipProxyConfig::CSipProxyConfig Constructor

Default Constructor.

C++

```
CSipProxyConfig();
```

3.5.1.1.2 - Methods**3.5.1.1.2.1 - CSipProxyConfig::AddRef Method**

Adds a reference to the object.

C++

```
unsigned int AddRef() const;
```

Returns

The CSipProxyConfig (see page 402)'s reference count after being increased.

Description

This method is used to increment the reference count number on a CSipProxyConfig (see page 402) instance. A reference is required when any object wants to keep a pointer to the CSipPacket (see page 447) after passing the CSipProxyConfig (see page 402) to another object, to ensure that the CSipProxyConfig (see page 402) object is not released from memory by another object.

See Also

Release (see page 405)

3.5.1.1.2.2 - GetAddresses**3.5.1.1.2.2.1 - CSipProxyConfig::GetAddresses Method**

Gets the addresses for which the Proxy is responsible.

C++

```
CVector<CHostPort>& GetAddresses();
```

Returns

The addresses.

Description

Gets the addresses for which the Proxy is responsible. The address at first index is used to populate the sent-by value in the top-most Via header. All addresses are compared against the maddr parameters in Request-URI and Via headers when the Proxy needs to make decisions.

Note that the port is important in the addresses. Only the address having EXACTLY the same address and port is matched. This means that inserting an address with the 5060 port does NOT match the same address having the 5062 port. Furthermore, an address having the 5060 port specified does NOT match an address having no port. To have an address matched to both 5060 and no port specified, both entries MUST be entered in the vector returned by this method. No port specified is ONLY equivalent to port 0.

Modifying the addresses once Proxy services have been created updates the values in every services sharing this object. The services keep a reference on this class and do not make a copy of the Proxy Configuration object.

3.5.1.1.2.2.2 - CSipProxyConfig::GetAddresses Method

Gets the addresses for which the Proxy is responsible.

C++

```
const CVector<CHostPort>& GetAddresses() const;
```

Returns

The addresses.

Description

Gets the addresses for which the Proxy is responsible. The address at first index is used to populate the sent-by value in the top-most Via header. All addresses are compared against the maddr parameters in Request-URI and Via headers when the Proxy needs to make decisions.

3.5.1.1.2.3 - GetAuthenticationRealm

3.5.1.1.2.3.1 - CSipProxyConfig::GetAuthenticationRealm Method

Gets the Realm that the proxy manages.

C++

```
CString& GetAuthenticationRealm();
```

Returns

The realm.

Description

Gets the realm managed by the proxy. This value is used when the proxy searches in the packet for a Proxy-Authorization header.

3.5.1.1.2.3.2 - CSipProxyConfig::GetAuthenticationRealm Method

Gets the Realm that the proxy manages.

C++

```
const CString& GetAuthenticationRealm() const;
```

Returns

The realm.

Description

Gets the realm managed by the proxy. This value is used when the proxy searches in the packet for a Proxy-Authorization header.

3.5.1.1.2.4 - GetProxyId

3.5.1.1.2.4.1 - CSipProxyConfig::GetProxyId Method

Gets the Proxy ID.

C++

```
CString& GetProxyId();
```

Returns

The salt value.

Description

Gets the ID used to identify a Proxy. If more than one Proxy resolves the same domain name, they all must use the same ID in order to recognize the packets processed by other proxies.

Modifying the ID once Proxy services have been created updates the value in every services sharing this object. The services keep a reference on this class and do not make a copy of the Proxy Configuration object.

3.5.1.1.2.4.2 - CSipProxyConfig::GetProxyId Method

Gets the Proxy ID.

C++

```
const CString& GetProxyId() const;
```

Returns

The salt value.

Description

Gets the ID used to identify a Proxy. If more than one Proxy resolves the same domain name, they all must use the same ID in order to recognize the packets processed by other proxies.

3.5.1.1.2.5 - CSipProxyConfig::Release Method

Releases a reference to the object.

C++

```
unsigned int Release() const;
```

Returns

The CSipProxyConfig (see page 402)'s reference count after being decreased.

Description

This method is used to decrement the CSipProxyConfig (see page 402)'s reference count. This needs to be called when the object owning a reference to the CSipProxyConfig (see page 402) no longer has any use for this CSipProxyConfig (see page 402). When the reference count reaches zero, the CSipProxyConfig (see page 402) deletes itself.

See Also

AddRef (see page 403)

3.5.1.2 - ISipSessionStatefulProxyMgr Class**Class Hierarchy**

 ISipSessionStatefulProxyMgr

C++

```
class ISipSessionStatefulProxyMgr;
```

Description

This is the interface through which the SIP session stateful proxy service reports event to its user.

If Record-Route headers are added to requests associated with the session, the initial INVITE incoming request that established the session is reported via the EvTargetFound (see page 410) or EvNoTargetFound (see page 406) event. All other incoming requests received within the session are reported through the event EvTargetFound (see page 410).

The manager notifies the application that the session has been established via the event EvSessionCreated (see page 409).

The session is terminated when the event EvSessionTerminated (see page 409) or EvTransactionFailed (see page 411) is called. EvSessionTerminated (see page 409) is called when the session has received a final response to a BYE request. EvTransactionFailed (see page 411) indicates a session termination when it is called for an initial INVITE request.

Location

SipProxy/ISipSessionStatefulProxyMgr.h

See Also

ISipSessionStatefulProxySvc (see page 414)

Methods

Method	Description
  EvMaxForwardReached (see page 406)	The request has transited through too many hops.

• A EvNoTargetFound (see page 406)	Reports to the application that a request cannot be forwarded according to current packet data.
• A EvRequestForwardTimeout (see page 408)	Reports to the application that the request being forwarded did not receive a final response within the allotted time.
• A EvSessionCreated (see page 409)	The session has been created because of 1xx or 2xx response reception.
• A EvSessionTerminated (see page 409)	Notifies the application that the session is terminated because a final response (>= 2xx) to BYE has been received.
• A EvTargetFound (see page 410)	Reports to the application that a request must be forwarded to a single target.
• A EvTransactionFailed (see page 411)	A final negative response (>= 300) has been received on a client transaction.
• A EvTransactionProgress (see page 412)	A provisional response has been received on a client transaction.
• A EvTransactionSucceeded (see page 413)	A 2xx response has been received on a client transaction.

Legend

•	Method
A	abstract

3.5.1.2.1 - Methods

3.5.1.2.1.1 - ISipSessionStatefulProxyMgr::EvMaxForwardReached Method

The request has transited through too many hops.

C++

```
virtual void EvMaxForwardReached(IN ISipSessionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rOriginalRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

Description

This event is reported to the manager when receiving a request with a Max-Forwards header that has a value of zero. This means the request must not be forwarded.

The manager should send a "483 Too Many Hops" response with the pServerEventCtrl.

3.5.1.2.1.2 - ISipSessionStatefulProxyMgr::EvNoTargetFound Method Updated behavior in 4.1.4

Reports to the application that a request cannot be forwarded according to current packet data.

C++

```
virtual void EvNoTargetFound(IN ISipSessionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality. This parameter can be NULL, which means that the request received is an ACK.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

IN CSipPacket& rUpdatedRequest	<p>The updated request to forward.</p> <p>Notice that this is a const parameter. As such, the manager must not modify this packet, but should instead create a copy of it and modify the copy before forwarding it. The manager still has to keep a reference to this packet by calling AddRef on it, otherwise it is deleted after this event call.</p> <p>The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet.</p> <p>Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.</p>
--------------------------------	--

Description

This event reports to the manager that the incoming request cannot be automatically forwarded. The manager must use any mechanism at its disposal (usually a location service) to find alternative destinations for this request. This event can be reported only once per session and it is for the initial INVITE request that established the session.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, on rUpdatedRequest, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check
- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter.
- Section 16.6 [5] - Optionally add additional header fields

- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the `lr` parameter). This set MUST be pushed into the `Route` header field of the copy ahead of any existing values, **if** present. If the `Route` header field is absent, it MUST be added, containing that list of URIs.

- Section 16.6 [8] - Update the topmost `Via` header field value with proper `Via` sent-by information, if required.

After finding suitable targets, the manager must create a new copy of `rUpdatedRequest` for each target to which it wants to forward the request and modify it according to the steps described previously.

The manager can call `ISipSessionStatefulProxySvc::ForwardRequest` (see page 417) once or multiple times. This has the effect of creating client transactions for forwarding the request. In this initial version, the manager must wait for a client transaction to terminate before trying the next target (serial forking) as parallel forking is not yet supported.

The manager can forward the request to as many destination as it wants. However, once a 2xx or 6xx response is received, the manager must no longer try to forward the request.

When the manager cannot find any suitable target to which forward the request, it must create an appropriate 4xx response packet to send to the peer. This can be done by either:

- Actually creating a response from the original request with the appropriate `CSipPacket` (see page 447) constructor and forward the response with `ISipSessionStatefulProxySvc::ForwardResponse` (see page 418). The response must not be created from `rUpdatedRequest`.
- Send the response with the provided `ISipServerEventControl` (see page 75).

See Also

`ISipSessionStatefulProxySvc::ForwardRequest` (see page 417), `ISipSessionStatefulProxySvc::ForwardResponse` (see page 418), `ISipServerEventControl` (see page 75)

3.5.1.2.1.3 - `ISipSessionStatefulProxyMgr::EvRequestForwardTimeout` Method

Reports to the application that the request being forwarded did not receive a final response within the allotted time.

C++

```
virtual void EvRequestForwardTimeout(IN ISipSessionStatefulProxySvc* pSvc, IN ISipClientTransaction* pClientTransaction) = 0;
```

Parameters

Parameters	Description
<code>IN ISipSessionStatefulProxySvc* pSvc</code>	The service that is reporting this event.
<code>IN ISipClientTransaction* pClientTransaction</code>	The interface to access client transaction related functionality.

Description

This event reports to the manager that a forwarded request was not answered with a final response within the allotted time, defined as the `uTimeoutMs` parameter of the `ISipSessionStatefulProxySvc::ForwardRequest` (see page 417) method.

Upon the reception of this event, an implementation could choose to cancel the currently outgoing request and try a different

destination, or it could choose to cancel the request and forward a final 408 response on the server transaction. Note however that the implementation should wait for a final response to be received before taking these actions, as cancelling a request does not always succeed.

Note that according to RFC 3261, only an INVITE transaction can be cancelled.

See Also

ISipSessionStatefulProxySvc::ForwardRequest (see page 417), ISipClientTransaction (see page 21)

3.5.1.2.1.4 - ISipSessionStatefulProxyMgr::EvSessionCreated Method

The session has been created because of 1xx or 2xx response reception.

C++

```
virtual void EvSessionCreated(IN ISipSessionStatefulProxySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CSipPacket& rOriginalResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

Description

Notifies the application that the session has been created following the reception of a provisional or final positive response.

The manager should call CallNextClientEvent on the pClientEventCtrl to obtain either the EvTransactionProgress (see page 412) or EvTransactionSucceeded (see page 413) event and appropriately forward the response.

3.5.1.2.1.5 - ISipSessionStatefulProxyMgr::EvSessionTerminated Method

Notifies the application that the session is terminated because a final response (>= 2xx) to BYE has been received.

C++

```
virtual void EvSessionTerminated(IN ISipSessionStatefulProxySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CSipPacket& rOriginalResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wishes to keep a reference to it after this event call, otherwise it may be deleted at any time.

Description

Tells the application that the session has terminated. The session is terminated when a final response to a BYE request is received. This is to allow the application to determine when to release the session stateful context. It is also terminated when a session was established by a provisional response to an initial INVITE and a final negative response is received for that INVITE.

The manager should call CallNextEvent on the pClientEventCtrl to appropriately forward the response.

See Also

ISipSessionStatefulProxyMgr::EvTransactionFailed (see page 411)

3.5.1.2.1.6 - ISipSessionStatefulProxyMgr::EvTargetFound Method Updated behavior in 4.1.4

Reports to the application that a request must be forwarded to a single target.

C++

```
virtual void EvTargetFound(IN ISipSessionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN
const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality. This parameter can be NULL, which means that the request received is an ACK.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedRequest	The updated request to forward. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call. The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet. Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.

Description

This event reports to the manager that the incoming request should be forwarded to a single target, as specified in RFC 3261. This event can be reported for the initial INVITE request that established the session or for all other requests within the session.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, on rUpdatedRequest, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check

- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter.
- Section 16.6 [5] - Optionally add additional header fields
- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field of the copy ahead of any existing values, **if** present. If the Route header field is absent, it MUST be added, containing that list of URIs.

- Section 16.6 [8] - Update the topmost Via header field value with proper Via sent-by information, if required.

After modifying rUpdatedRequest as described in the above steps, the manager should forward the request to the proper destination by using the ISipSessionStatefulProxySvc::ForwardRequest (see page 417) API.

See Also

ISipSessionStatefulProxySvc::ForwardRequest (see page 417)

3.5.1.2.1.7 - ISipSessionStatefulProxyMgr::EvTransactionFailed Method

A final negative response (>= 300) has been received on a client transaction.

C++

```
virtual void EvTransactionFailed(IN ISipSessionStatefulProxySvc* pSvc, IN mxt_opaque opqTransactionId, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN mxt_opaque opqTransactionId	The opaque transaction id that must be passed when forwarding the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

IN CSipPacket& rUpdatedResponse	The negative response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.
---------------------------------	---

Description

This event reports to the manager that a failure response has been received on a client transaction. When it is called for an initial INVITE request, it indicates that the session has terminated.

rUpdatedResponse is not necessarily the exact same response that was received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context
- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [4] - Add response to context (save it as a possible response to forward).
- Section 16.7 [6] - Choosing best response, if all client branches have failed.
- Section 16.7 [7] - Aggregate Authorization headers, if forwarding a 401 or 407 response.
- Section 16.7 [8] - Optionally rewrite Record-Route header field values.
- Section 16.7 [9] - Forward the response if necessary
- Section 16.7 [10] - Generate any necessary CANCEL requests (when receiving a 6xx class response).

The manager uses the ISipSessionStatefulProxySvc::ForwardResponse (see page 418) API to forward the response downstream.

See Also

ISipSessionStatefulProxySvc::ForwardResponse (see page 418), EvTransactionProgress (see page 412), EvTransactionSucceeded (see page 413)

3.5.1.2.1.8 - ISipSessionStatefulProxyMgr::EvTransactionProgress Method

A provisional response has been received on a client transaction.

C++

```
virtual void EvTransactionProgress(IN ISipSessionStatefulProxySvc* pSvc, IN mxt_opaque opqTransactionId, IN
ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket&
rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN mxt_opaque opqTransactionId	The opaque transaction id that must be passed when forwarding the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wishes to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The provisional response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a provisional response has been received on a client transaction.

rUpdatedResponse is not necessarily the exact same response that was received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context
- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [5] - Check response for forwarding. All provisional responses except for 100 Trying responses must immediately be forwarded, unless a 2xx response was already forwarded.
- Section 16.7 [8] - Optionally rewrite Record-Route header field values.
- Section 16.7 [9] - Forward the response

The manager uses the ISipSessionStatefulProxySvc::ForwardResponse (see page 418) API to forward the response downstream.

See Also

ISipSessionStatefulProxySvc::ForwardResponse (see page 418), EvTransactionSucceeded (see page 413), EvTransactionFailed (see page 411)

3.5.1.2.1.9 - ISipSessionStatefulProxyMgr::EvTransactionSucceeded Method

A 2xx response has been received on a client transaction.

C++

```
virtual void EvTransactionSucceeded(IN ISipSessionStatefulProxySvc* pSvc, IN mxt_opaque opqTransactionId, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxySvc* pSvc	The service that is reporting this event.
IN mxt_opaque opqTransactionId	The opaque transaction id that must be passed when forwarding the response.

IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The 2xx response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a 2xx response has been received on a client transaction.

rUpdatedResponse is not necessarily the exact same response that was received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context
- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [5] - Check response for forwarding. All 2xx responses from all client transactions must be forwarded immediately.
- Section 16.7 [8] - Optionally rewrite Record-Route header field values.
- Section 16.7 [9] - Forward the response
- Section 16.7 [10] - Cancel remaining client transactions

The manager uses the ISipSessionStatefulProxySvc::ForwardResponse (see page 418) API to forward the response downstream.

See Also

ISipSessionStatefulProxySvc::ForwardResponse (see page 418), EvTransactionProgress (see page 412), EvTransactionFailed (see page 411)

3.5.1.3 - ISipSessionStatefulProxySvc Class

Class Hierarchy



C++

```
class ISipSessionStatefulProxySvc : public IEComUnknown;
```

Description

The session stateful proxy service allows the application to forward requests and responses received on the same session created for an initial INVITE request.

It does not allow to forward a received INVITE to multiple targets at the same time, known as parallel forking. However, it supports downstream parallel forking, which means it will behave properly when a proxy further down the path does parallel forking. Hence, the application that needs to act as a session stateful proxy and needs to parallel fork the INVITE request simply needs to forward the INVITE to itself and manage it with another context to which the ISipTransactionStatefulProxySvc (see page 435) is attached and

perform the parallel forking on that context.

To use this service and see every requests associated with the session, the application must add Record-Route headers to requests that are part of the session. For example, not doing so would make it impossible for ACK requests to go through the Proxy.

The following are the roles and responsibilities of this service:

- Detect 2xx responses to INVITE retransmissions and silently forward them.
- Match INVITE related requests and responses (reINVITE, CANCEL, ACK, and BYE) to the session.
- Match UPDATE and PRACK requests and responses to the session.

• Report to the application that a session is terminated. This is the case when a final negative response to an initial INVITE request or when a final response to BYE request is received.

- Detect new early sessions (1xx response with different to-tag).
- Detect new sessions (2xx response with different to-tag).

Please note that if a call lasts for less than the timeout timer, there are some forwardable packets that stay alive after the final response to BYE. By clearing the context, the forwardable packets are deleted and retransmission to 1xx, 2xx (to initial INVITE), or ACK are ignored.

The ISipSessionStatefulProxySvc is an ECOM object

The ISipSessionStatefulProxySvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipSessionStatefulProxySvc

Interface Id: IID_ISipSessionStatefulProxySvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipProxy/ISipSessionStatefulProxySvc.h

See Also

ISipSessionStatefulProxyMgr (see page 405)

Methods

Method	Description
•   AddLocalRecordRoute (see page 416)	Creates a Record-Route header and adds it to the request. The host and port are later overridden based on the actual network interface and port used to send the packet.
•   AddRecordRoute (see page 416)	Creates a Record-Route header and adds it to the request.
•   CancelAllClientTransactions (see page 417)	Sends a CANCEL on all non-terminated client transactions.
•   ForwardRequest (see page 417)	Forwards a request.
•   ForwardResponse (see page 418)	Sends or forwards a response on the server transaction.
•   SetConfig (see page 419)	Sets the configuration object used by the stateful service.
•   SetForkedDialogGrouperMgr (see page 419)	Configures the forked dialog grouper manager that will be informed when supplemental dialogs are needed.

 SetManager (see page 420)	Configures the manager that receives events from this service.
---	--

Legend

	Method
	abstract

3.5.1.3.1 - Methods

3.5.1.3.1.1 - ISipSessionStatefulProxySvc::AddLocalRecordRoute Method New in 4.1.4

Creates a Record-Route header and adds it to the request. The host and port are later overriden based on the actual network interface and port used to send the packet.

C++

```
virtual mxt_result AddLocalRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rRecordRouteNameAddr	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and the username is overriden. It must be either a SIP or a SIPS URI. The host/port portion is later overriden to reflect the actual network interface and port used to send the packet.
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

Returns

- resS_OK: The Record-Route has been added.
- resFE_INVALID_ARGUMENT: The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- resFE_FAIL: The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteNameAddr parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (it adds it if not present). It also creates a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity. Finally, it adds this Record-Route to the request in parameter.

This method differs from AddRecordRoute (see page 416) in that the host/port from rRecordRouteNameAddr is discarded and later replaced by the actual interface and port used to send the request.

See Also

ForwardRequest (see page 417), AddRecordRoute (see page 416)

3.5.1.3.1.2 - ISipSessionStatefulProxySvc::AddRecordRoute Method

Creates a Record-Route header and adds it to the request.

C++

```
virtual mxt_result AddRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rRecordRouteNameAddr	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and the username is overridden. It must be either a SIP or a SIPS URI.
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

Returns

- resS_OK: The Record-Route has been added.
- resFE_INVALID_ARGUMENT: The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- resFE_FAIL: The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteNameAddr parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (it adds it if not present). It also creates a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity.

Finally, it adds this Record-Route to the request in parameter.

See Also

ForwardRequest (see page 417), AddLocalRecordRoute (see page 416)

3.5.1.3.1.3 - ISipSessionStatefulProxySvc::CancelAllClientTransactions Method

Sends a CANCEL on all non-terminated client transactions.

C++

```
virtual mxt_result CancelAllClientTransactions(IN ISipServerEventControl* pServerEventCtrl) = 0;
```

Parameters

Parameters	Description
IN ISipServerEventControl* pServerEventCtrl	The server event control for which to cancel all client transactions.

Returns

- resS_OK: Success.
- resFE_INVALID_STATE: The object was not configured with its manager.
- resFE_INVALID_ARGUMENT: The pServerEventCtrl does not match any server transaction currently managed by this service.

Description

Sends a CANCEL on all non-terminated client transactions corresponding to the given server transaction.

3.5.1.3.1.4 - ISipSessionStatefulProxySvc::ForwardRequest Method

Forwards a request.

C++

```
virtual mxt_result ForwardRequest(IN CSipPacket& rRequest, IN mxt_opaque opqApplication, IN
ISipServerEventControl* pServerEventCtrl, OUT ISipClientTransaction*& rpTransaction, IN uint64_t uTimeoutMs = 0)
= 0;
```

Parameters

Parameters	Description
IN CSipPacket& rRequest	The request to forward. The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454) on this parameter, once for each reference the caller has to this packet. This packet is used and modified by the stack before actually forwarding it.
IN mxt_opaque opqApplication	Application-provided opaque identifier for the client transaction that is created to forward the request. This is not used by the stack, but it is instead reported as-is in the various events concerning this new transaction.
IN ISipServerEventControl* pServerEventCtrl	The incoming server transaction, this is the parameter passed in the EvTargetFound or EvNoTargetFound event. This parameter is used by the session stateful proxy service to retrieve the transaction on which the request was received.
OUT ISipClientTransaction*& rpTransaction	When this method returns success, this parameter holds the transaction interface which offers some more functionality related to the newly created client transaction. Otherwise, it is NULL. When non-NULL, a reference is counted on that parameter and the reference is under the responsibility of the caller of the method.
IN uint64_t uTimeoutMs = 0	Optional. The time, in milliseconds, after the EvRequestForwardTimeout event is reported to the manager when no final response to that forwarded request has been received. When the uTimeoutMs is 0, the timer is not started, i.e., the EvRequestForwardTimeout event is not reported. The default uTimeoutMs value is 0.

Returns

- resS_OK: Success.
- resFE_INVALID_STATE: The object was not configured with its manager or a request has not yet been received.
- resFE_INVALID_ARGUMENT: The packet is a response or the method is non-ACK and the transaction can not be found.
- failure code: General failure cases.

Description

This method is used by the manager to create a new client transaction and forward on it a request it has received through the ISipTransactionStatefulProxyMgr::EvTargetFound (see page 431) or ISipTransactionStatefulProxyMgr::EvNoTargetFound (see page 429) events.

As described in the ISipSessionStatefulProxyMgr (see page 405) interface, the manager must properly update the SIP request before forwarding it. This includes updating the request-URI (if necessary) and adding Record-Route headers.

The stack will later use the ISipSessionStatefulProxyMgr interface to report the event about the newly created client transaction. This specific transactionOpaque parameter is provided back to the manager when reporting events for this client transaction.

See Also

ISipSessionStatefulProxyMgr::EvTargetFound (see page 410), ISipSessionStatefulProxyMgr::EvNoTargetFound (see page 406)

3.5.1.3.1.5 - ISipSessionStatefulProxySvc::ForwardResponse Method

Sends or forwards a response on the server transaction.

C++

```
virtual mxt_result ForwardResponse(IN CSipPacket& rResponse, IN mxt_opaque opqTransactionId) = 0;
```

Parameters

Parameters	Description
IN CSipPacket& rResponse	The response to forward. The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454()) on this parameter, once for each reference the caller has to this packet. This packet is used and modified by the stack before actually forwarding it.
IN mxt_opaque opqTransactionId	The opaque transaction identifier, this is the parameter passed in EvTransactionProgress, EvTransactionSucceeded or EvTransactionFailed event. This parameter is used by the session stateful proxy service to retrieve the transaction on which the response was received.

Returns

- resS_OK: Success.
- resFE_INVALID_STATE: The object was not configured with its manager.
- failure code: General failure cases.

Description

This method is used only to forward a response from a client transaction on the server transaction. To create its own response from the original request to send, the server must call SendResponse on the server event control instead.

When forwarding a 2xx response, this service automatically cancels all remaining client branches.

This method can be used to answer requests that are specifically for this device, like a REGISTER request for this device's domain.

See Also

ISipSessionStatefulProxyMgr::EvTransactionProgress (see page 412), ISipSessionStatefulProxyMgr::EvTransactionSucceeded (see page 413), ISipSessionStatefulProxyMgr::EvTransactionFailed (see page 411), ISipSessionStatefulProxyMgr::EvSessionTerminated (see page 409)

3.5.1.3.1.6 - ISipSessionStatefulProxySvc::SetConfig Method

Sets the configuration object used by the stateful service.

C++

```
virtual void SetConfig(IN const CSipProxyConfig& rConfig) = 0;
```

Parameters

Parameters	Description
IN const CSipProxyConfig& rConfig	The proxy configuration object.

Description

Sets the configuration object for this proxy service. It is recommended to call this method only once at service creation time to avoid potential problems when the configuration changes at runtime.

3.5.1.3.1.7 - ISipSessionStatefulProxySvc::SetForkedDialogGrouperMgr Method

Configures the forked dialog grouper manager that will be informed when supplemental dialogs are needed.

C++

```
virtual mxt_result SetForkedDialogGrouperMgr(IN ISipForkedDialogGrouperMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipForkedDialogGrouperMgr* pMgr	Pointer to the manager. It can be NULL, which means that this service does not support forking. The behaviour is then undefined when receiving forked responses as in previous versions.

Returns

- resS_OK: Manager configured successfully.
- resFE_INVALID_STATE: The manager is already set.

Description

Configures the manager that will be informed when supplemental dialogs are needed. Once a non NULL manager has been set, it cannot be changed, not even to put it back to NULL. It must remain valid for the entire life of the service.

3.5.1.3.1.8 - ISipSessionStatefulProxySvc::SetManager Method

Configures the manager that receives events from this service.

C++

```
virtual mxt_result SetManager(IN ISipSessionStatefulProxyMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipSessionStatefulProxyMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

- resS_OK: Manager configured successfully.
- resFE_INVALID_ARGUMENT: NULL pointer provided as parameter.

Description

Configures the manager that receives events from this service.

3.5.1.4 - ISipStatelessProxyMgr Class**Class Hierarchy**

ISipStatelessProxyMgr

C++

```
class ISipStatelessProxyMgr;
```

Description

This is the interface through which the SIP transaction stateless proxy service reports event to its user.

Location

SipProxy/ISipStatelessProxyMgr.h

See Also

ISipStatelessProxySvc (see page 425)

Methods

Method	Description
◆ EvMaxForwardReached (see page 421)	The request has transited through too many hops.

• A EvNoTargetFound (see page 421)	Reports to the application that a request cannot be forwarded according to current packet data.
• A EvResponseReceived (see page 423)	A response has been received.
• A EvTargetFound (see page 423)	Reports to the manager that a request must be forwarded to a single target.

Legend

•	Method
A	abstract

3.5.1.4.1 - Methods

3.5.1.4.1.1 - ISipStatelessProxyMgr::EvMaxForwardReached Method

The request has transited through too many hops.

C++

```
virtual void EvMaxForwardReached(IN ISipStatelessProxySvc* pSvc, IN const CSipPacket& rOriginalRequest) = 0;
```

Parameters

Parameters	Description
IN ISipStatelessProxySvc* pSvc	The service that is reporting this event.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

Description

This event is reported to the manager when receiving a request with a Max-Forwards header that has a value of zero. This means the request must not be forwarded.

The manager should send a "483 Too Many Hops" response.

This can be done by actually creating a response from the original request with the appropriate CSipPacket (see page 447) constructor and forwarding the response with ISipStatelessProxySvc::ForwardResponse (see page 427).

See Also

ISipStatelessProxySvc::ForwardResponse (see page 427)

3.5.1.4.1.2 - ISipStatelessProxyMgr::EvNoTargetFound Method Updated behavior in 4.1.4

Reports to the application that a request cannot be forwarded according to current packet data.

C++

```
virtual void EvNoTargetFound(IN ISipStatelessProxySvc* pSvc, IN const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipStatelessProxySvc* pSvc	The service that is reporting this event.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

IN CSipPacket& rUpdatedRequest	<p>The updated request to forward.</p> <p>Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.</p> <p>The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet.</p> <p>Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.</p>
--------------------------------	--

Description

This event reports to the manager that the incoming request cannot be automatically forwarded. The manager must use any mechanism at its disposal (usually a location service) to find a single alternative destination for this request. Retransmissions of this request must yield the same destination.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check
- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter
- Section 16.6 [5] - Optionally add additional header fields
- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field of the copy ahead of any existing values, **if** present. If the Route header field is absent, it MUST be added, containing that list of URIs.

- Section 16.6 [8] - Update the topmost Via header field value with proper Via sent-by information, if required.

After finding the target, the manager must modify rUpdatedRequest according to the steps described previously and forward it.

The manager can call ISipStatelessProxySvc::ForwardRequest (see page 427) only once for this packet. A stateless proxy forwards packets only once and immediately forgets about them.

When the manager cannot find any suitable target to which forward the request, it must create an appropriate 4xx response packet to send to the peer. This can be done by actually creating a response from the original request with the appropriate CSipPacket (see

page 447) constructor and forwarding the response with ISipStatelessProxySvc::ForwardResponse (see page 427). The response must not be created from rUpdatedRequest.

See Also

ISipStatelessProxySvc::ForwardRequest (see page 427), ISipStatelessProxySvc::ForwardResponse (see page 427)

3.5.1.4.1.3 - ISipStatelessProxyMgr::EvResponseReceived Method

A response has been received.

C++

```
virtual void EvResponseReceived(IN ISipStatelessProxySvc* pSvc, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipStatelessProxySvc* pSvc	The service that is reporting this event.
IN const CSipPacket& rOriginalResponse	The original received response, unmodified. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The response. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a response has been received on the stateless service.

rResponse is not necessarily the exact same response that was received by the manager. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [8] - Optionally rewrite Record-Route header field values
- Section 16.7 [9] - Forward the response

The manager uses the ISipStatelessProxySvc::ForwardResponse (see page 427) API to forward the response downstream.

See Also

ISipStatelessProxySvc::ForwardResponse (see page 427),

3.5.1.4.1.4 - ISipStatelessProxyMgr::EvTargetFound Method Updated behavior in 4.1.4

Reports to the manager that a request must be forwarded to a single target.

C++

```
virtual void EvTargetFound(IN ISipStatelessProxySvc* pSvc, IN const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipStatelessProxySvc* pSvc	The service that is reporting this event.

IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedRequest	The updated request to forward. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call. The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet. Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.

Description

This event reports to the manager that the incoming request should be forwarded to a single target, as specified in RFC 3261.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check
- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter
- Section 16.6 [5] - Optionally add additional header fields
- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field of the copy ahead of any existing values, **if** present. If the Route header field is absent, it MUST be added, containing that list of URIs.

- Section 16.6 [8] - Update the topmost Via header field value with proper Via sent-by information, if required.

After modifying the packet as described in the above steps, the manager should forward the request to the proper destination by using the ISipStatelessProxySvc::ForwardRequest (see page 427) API.

See Also

[ISipStatelessProxySvc::ForwardRequest](#) (see page 427)

3.5.1.5 - ISipStatelessProxySvc Class**Class Hierarchy****C++**

```
class ISipStatelessProxySvc : public IEComUnknown;
```

Description

This service offers basic services used to create a stateless SIP proxy implementation as defined in RFC 3261. It offers a lot of flexibility, mainly to allow the creation of firewall proxy servers and servers that sit on a NAT.

This service offers the following:

- Allows adding Record-Route headers to outgoing transactions.
- Allows complete control of forwarded requests and responses.
- Allows the application to form its own failure response.
- Automatically updates the Via of all outgoing transactions.

The ISipStatelessProxySvc is an ECOM object

The ISipStatelessProxySvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipStatelessProxySvc

Interface Id: IID_ISipStatelessProxySvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipProxy/ISipStatelessProxySvc.h

See Also

[ISipTransactionStatefulProxyMgr](#) (see page 428)

Methods

Method	Description
• A AddLocalRecordRoute (see page 426)	Creates a Record-Route header and adds it to the request. The host and port are later overridden based on the actual network interface and port used to send the packet.
• A AddRecordRoute (see page 426)	Creates a Record-Route header and adds it to the request.
• A ForwardRequest (see page 427)	Forwards a request.
• A ForwardResponse (see page 427)	Sends or forwards a response.
• A SetConfig (see page 428)	Sets the configuration object used by the stateless service.
• A SetManager (see page 428)	Configures the manager that receives events from this service.

Legend

	Method
	abstract

3.5.1.5.1 - Methods**3.5.1.5.1.1 - ISipStatelessProxySvc::AddLocalRecordRoute Method New in 4.1.4**

Creates a Record-Route header and adds it to the request. The host and port are later overriden based on the actual network interface and port used to send the packet.

C++

```
virtual mxt_result AddLocalRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rRecordRouteNameAddr	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and the username is overriden. It must be either a SIP or a SIPS URI. The host/port portion is later overriden to reflect the actual network interface and port used to send the packet.
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

Returns

- resS_OK: The Record-Route has been added.
- resFE_INVALID_ARGUMENT: The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- resFE_FAIL: The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteNameAddr parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (it adds it if not present). It also creates a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity. Finally, it adds this Record-Route to the request in parameter.

This method differs from AddRecordRoute (see page 426) in that the host/port from rRecordRouteNameAddr is discarded and later replaced by the actual interface and port used to send the request.

See Also

ForwardRequest (see page 427), AddRecordRoute (see page 426)

3.5.1.5.1.2 - ISipStatelessProxySvc::AddRecordRoute Method

Creates a Record-Route header and adds it to the request.

C++

```
virtual mxt_result AddRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

rRecordRouteUri	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and the username is overridden.
-----------------	---

Returns

- resS_OK: The Record-Route has been added.
- resFE_INVALID_ARGUMENT: The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- resFE_FAIL: The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteNameAddr parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (it adds it if not present). It also creates a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity.

Finally, it adds this Record-Route to the request in parameter.

See Also

ForwardRequest (see page 427)

3.5.1.5.1.3 - ISipStatelessProxySvc::ForwardRequest Method

Forwards a request.

C++

```
virtual mxt_result ForwardRequest(IN CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN CSipPacket& rRequest	<p>The request to forward.</p> <p>The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454()) on this parameter, once for each reference the caller has to this packet.</p> <p>This packet is used and modified by the stack before actually forwarding it.</p>

Returns

- Success or failure code.

Description

This method is used by the manager to statelessly forward a request it has received through the ISipTransactionStatefulProxyMgr::EvTargetFound (see page 431) and ISipTransactionStatefulProxyMgr::EvNoTargetFound (see page 429) events.

As described in the ISipStatelessProxyMgr (see page 420) interfaces, the manager must properly update the SIP request before forwarding it. This includes updating the request-URI (if necessary) and adding Record-Route headers.

No events are reported back by the service about this packet once it is forwarded.

See Also

ISipStatelessProxyMgr::EvTargetFound (see page 423), ISipStatelessProxyMgr::EvNoTargetFound (see page 421)

3.5.1.5.1.4 - ISipStatelessProxySvc::ForwardResponse Method

Sends or forwards a response.

C++

```
virtual mxt_result ForwardResponse(IN CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN CSipPacket& rResponse	<p>The response to forward.</p> <p>The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454()) on this parameter, once for each reference the caller has to this packet.</p> <p>This packet is used and modified by the stack before actually forwarding it.</p>

Returns

- Success or failure code.

Description

This method is used to statelessly send a response.

3.5.1.5.1.5 - ISipStatelessProxySvc::SetConfig Method

Sets the configuration object used by the stateless service.

C++

```
virtual void SetConfig(IN const CSipProxyConfig& rConfig) = 0;
```

Parameters

Parameters	Description
IN const CSipProxyConfig& rConfig	The proxy configuration object.

Description

Sets the configuration object for this proxy service. It is recommended to call this method only once at service creation time to avoid potential problems when the configuration changes at runtime.

3.5.1.5.1.6 - ISipStatelessProxySvc::SetManager Method

Configures the manager that receives events from this service.

C++

```
virtual mxt_result SetManager(IN ISipStatelessProxyMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipStatelessProxyMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

- resS_OK: Manager configured successfully.
- resFE_INVALID_ARGUMENT: NULL pointer provided as parameter.

Description

Configures the manager that receives events from this service.

3.5.1.6 - ISipTransactionStatefulProxyMgr Class**Class Hierarchy**

ISipTransactionStatefulProxyMgr

C++

```
class ISipTransactionStatefulProxyMgr;
```

Description

This is the interface through which the SIP transaction stateful proxy service reports an event to its user.

Location

SipProxy/ISipTransactionStatefulProxyMgr.h

See Also

ISipTransactionStatefulProxySvc (see page 435)

Methods

Method	Description
• A EvMaxForwardReached (see page 429)	The request has transited through too many hops.
• A EvNoTargetFound (see page 429)	Reports to the application that a request cannot be forwarded according to current packet data.
• A EvRequestForwardTimeout (see page 431)	Reports to the application that the request being forwarded did not receive a final response within the allotted time.
• A EvTargetFound (see page 431)	Reports to the application that a request must be forwarded to a single target.
• A EvTransactionFailed (see page 433)	A final negative response (>= 300) has been received on a client transaction.
• A EvTransactionProgress (see page 434)	A provisional response has been received on a client transaction.
• A EvTransactionSucceeded (see page 435)	A 2xx response has been received on a client transaction.

Legend

•	Method
A	abstract

3.5.1.6.1 - Methods**3.5.1.6.1.1 - ISipTransactionStatefulProxyMgr::EvMaxForwardReached Method**

The request has transited through too many hops.

C++

```
virtual void EvMaxForwardReached(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rOriginalRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.

Description

This event is reported to the manager when receiving a request with a Max-Forwards header that has a value of zero. This means the request must not be forwarded.

The manager should send a "483 Too Many Hops" response with the pServerEventCtrl.

3.5.1.6.1.2 - ISipTransactionStatefulProxyMgr::EvNoTargetFound Method Updated behavior in 4.1.4

Reports to the application that a request cannot be forwarded according to current packet data.

C++

```
virtual void EvNoTargetFound(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedRequest	The updated request to forward. Notice that this is a const parameter. As such, the manager must not modify this packet, but should instead create a copy of it and modify the copy before forwarding it. The manager still has to keep a reference to this packet by calling AddRef on it, otherwise it is deleted after this event call. The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet. Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.

Description

This event reports to the manager that the incoming request cannot be automatically forwarded. The manager must use any mechanism at its disposal (usually a location service) to find alternative destinations for this request.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, on rUpdatedRequest, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check
- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter
- Section 16.6 [5] - Optionally add additional header fields
- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field

of the copy ahead of any existing values, **if** present. If the Route header field is absent, it MUST be added, containing that list of URIs.

- Section 16.6 [8] - Update the topmost Via header field value with proper Via sent-by information, if required.

After finding suitable targets, the manager must create a new copy of rUpdatedRequest for each target to which it wants to forward the request and modify it according to the steps described previously.

The manager can call ISipTransactionStatefulProxySvc::ForwardRequest (see page 438) once or multiple times. This has the effect of creating client transactions for forwarding the request. The manager can either wait for a client transaction to terminate before trying the next target (serial forking) or it can choose to simultaneously forward the request to multiple targets at the same time (parallel forking).

The manager can forward the request to as many destinations as it wants. However, once a 2xx or 6xx response is received, the manager must no longer try to forward the request.

When the manager cannot find any suitable target to which forward the request, it must create an appropriate 4xx response packet to send to the peer. This can be done by either:

- Actually creating a response from the original request with the appropriate CSipPacket (see page 447) constructor and forward the response with ISipTransactionStatefulProxySvc::ForwardResponse (see page 439). The response must not be created from rUpdatedRequest.
- Send the response with the provided ISipServerEventControl (see page 75).

See Also

ISipTransactionStatefulProxySvc::ForwardRequest (see page 438), ISipTransactionStatefulProxySvc::ForwardResponse (see page 439), ISipServerEventControl (see page 75)

3.5.1.6.1.3 - ISipTransactionStatefulProxyMgr::EvRequestForwardTimeout Method

Reports to the application that the request being forwarded did not receive a final response within the allotted time.

C++

```
virtual void EvRequestForwardTimeout(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipClientTransaction* pClientTransaction) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientTransaction* pClientTransaction	The interface to access client transaction related functionality.

Description

This event reports to the manager that a forwarded request was not answered with a final response within the allotted time, defined as the uTimeoutMs parameter of the ISipTransactionStatefulProxySvc::ForwardRequest (see page 438) method.

Upon the reception of this event, an implementation could choose to cancel the currently outgoing request and try a different destination, or it could choose to cancel the request and forward a final 408 response on the server transaction. Note however that the implementation should wait for a final response to be received before taking these actions, as cancelling a request does not always succeed.

Note that according to RFC 3261, only an INVITE transaction can be cancelled.

See Also

ISipTransactionStatefulProxySvc::ForwardRequest (see page 438), ISipClientTransaction (see page 21)

3.5.1.6.1.4 - ISipTransactionStatefulProxyMgr::EvTargetFound Method Updated behavior in 4.1.4

Reports to the application that a request must be forwarded to a single target.

C++

```
virtual void EvTargetFound(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rOriginalRequest, IN CSipPacket& rUpdatedRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipServerEventControl* pServerEventCtrl	The interface to access server transaction related functionality.
IN const CSipPacket& rOriginalRequest	The original received request. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedRequest	The updated request to forward. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call. The updated request contains a top Via header added by the stack and the application is free to add parameters to this header before forwarding the packet. Updated Behavior: The Sent-By parameter contained in the top Via of the updated request is tentative and may be overridden when forwarding.

Description

This event reports to the manager that the incoming request should be forwarded to a single target, as specified in RFC 3261.

rUpdatedRequest can be seen as a pre-processed packet that can be used as a template to forward the request. The packet has been processed with the following steps from RFC 3261, when applicable:

- Section 16.3 [1] - Reasonable syntax check
- Section 16.3 [3] - Max-Forwards check
- Section 16.4 - Route Information Preprocessing
- Section 16.5 - Determining Request Targets
- Section 16.6 [3] - Max-Forwards update
- Section 16.6 [8] - Add a Via header field value.
- Remove, when found, a Proxy-Authorization header that corresponds to the realm managed by the proxy.

The manager's responsibilities upon receiving this event are as follows, on rUpdatedRequest, again according to RFC 3261:

- Section 16.3 [2] - URI scheme check
- Section 16.3 [5] - Proxy-Require check
- Section 16.6 [1] - Make a copy of the received request
- Section 16.6 [2] - Update the Request-URI
- Section 16.6 [4] - Optionally add a Record-route header field value with ;lr parameter
- Section 16.6 [5] - Optionally add additional header fields
- Section 16.6 [6] - Postprocess routing information, only with regards to inserting Routes to force the request to visit a specific set of proxies, as explained in the following excerpt from RFC 3261:

A proxy MAY have a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination. A proxy MUST ensure that all such proxies are loose routers. Generally, **this** can only be known with certainty **if** the proxies are within the same administrative domain. This set of proxies is represented by a set of URIs (each of which contains the lr parameter). This set MUST be pushed into the Route header field of the copy ahead of any existing values, **if** present. If the Route header field is absent, it MUST be added, containing that list of

URIs.

- Section 16.6 [8] - Update the topmost Via header field value with proper Via sent-by information, if required.

After modifying rUpdatedRequest as described in the above steps, the manager should forward the request to the proper destination by using the ISipTransactionStatefulProxySvc::ForwardRequest (see page 438) API.

See Also

ISipTransactionStatefulProxySvc::ForwardRequest (see page 438)

3.5.1.6.1.5 - ISipTransactionStatefulProxyMgr::EvTransactionFailed Method

A final negative response (>= 300) has been received on a client transaction.

C++

```
virtual void EvTransactionFailed(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The negative response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a failure response has been received on a client transaction.

rResponse is not necessarily the exact same response that was received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context.
- Section 16.7 [3] - Remove the topmost Via.

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [4] - Add response to context (save it as a possible response to forward).
- Section 16.7 [6] - Choosing best response, if all client branches have failed.
- Section 16.7 [7] - Aggregate Authorization headers, if forwarding a 401 or 407 response.

- Section 16.7 [8] - Optionally rewrite Record-Route header field values.
- Section 16.7 [9] - Forward the response if necessary.
- Section 16.7 [10] - Generate any necessary CANCEL requests (when receiving a 6xx class response).

The manager uses the `ISipTransactionStatefulProxySvc::ForwardResponse` (see page 439) API to forward the response downstream.

See Also

`ISipTransactionStatefulProxySvc::ForwardResponse` (see page 439), `EvTransactionProgress` (see page 434), `EvTransactionSucceeded` (see page 435)

3.5.1.6.1.6 - `ISipTransactionStatefulProxyMgr::EvTransactionProgress` Method

A provisional response has been received on a client transaction.

C++

```
virtual void EvTransactionProgress(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call <code>CSipPacket::AddRef</code> (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The provisional response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call <code>CSipPacket::AddRef</code> (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a provisional response has been received on a client transaction.

`rResponse` is not necessarily the exact same response that has been received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context
- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [5] - Check response for forwarding. All provisional responses except for 100 Trying responses must immediately be forwarded, unless a 2xx response was already forwarded.
- Section 16.7 [8] - Optionally rewrite Record-Route header field values
- Section 16.7 [9] - Forward the response

The manager uses the `ISipTransactionStatefulProxySvc::ForwardResponse` (see page 439) API to forward the response downstream.

See Also

`ISipTransactionStatefulProxySvc::ForwardResponse` (see page 439), `EvTransactionSucceeded` (see page 435), `EvTransactionFailed`

([see page 433](#))

3.5.1.6.1.7 - ISipTransactionStatefulProxyMgr::EvTransactionSucceeded Method

A 2xx response has been received on a client transaction.

C++

```
virtual void EvTransactionSucceeded(IN ISipTransactionStatefulProxySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rOriginalResponse, IN CSipPacket& rUpdatedResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxySvc* pSvc	The service that is reporting this event.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client transaction related functionality.
IN const CSipPacket& rOriginalResponse	The original received response. The manager should call CSipPacket::AddRef (see page 451) on this packet if it wants to keep a reference to it after this event call, otherwise it may be deleted at any time.
IN CSipPacket& rUpdatedResponse	The 2xx response, updated by this service. Notice that this is not a const parameter. As such, the manager is entitled to modify the packet before forwarding it. The manager that keeps a reference on this packet beyond the execution of this event call must call CSipPacket::AddRef (see page 451) on it, otherwise it is deleted after this event call.

Description

This event reports to the manager that a 2xx response has been received on a client transaction.

rResponse is not necessarily the exact same response that has been received on the client transaction. It was pre-processed with the following items from RFC 3261:

- Section 16.7 [1] - Find the appropriate response context
- Section 16.7 [3] - Remove the topmost Via

Upon receiving this response, the manager must perform the following items, again from RFC 3261:

- Section 16.7 [5] - Check response for forwarding. All 2xx responses from all client transactions must be forwarded immediately.
- Section 16.7 [8] - Optionally rewrite Record-Route header field values
- Section 16.7 [9] - Forward the response
- Section 16.7 [10] - Cancel remaining client transactions

The manager uses the ISipTransactionStatefulProxySvc::ForwardResponse ([see page 439](#)) API to forward the response downstream.

See Also

ISipTransactionStatefulProxySvc::ForwardResponse ([see page 439](#)), EvTransactionProgress ([see page 434](#)), EvTransactionFailed ([see page 433](#))

3.5.1.7 - ISipTransactionStatefulProxySvc Class

Class Hierarchy



C++

```
class ISipTransactionStatefulProxySvc : public IEComUnknown;
```

Description

This service offers basic services used to create a stateful SIP proxy implementation as defined in RFC 3261. It offers a lot of flexibility, mainly to allow the creation of firewall proxy servers and servers that sit on a NAT.

This service offers the following:

- Allows serial and parallel forking along with recursive searches.
- Allows adding Record-Route headers to outgoing transactions.
- Allows complete control of forwarded requests and responses.
- Allows the application to form its own failure response.
- Automatically manages incoming CANCEL requests by cancelling outstanding transactions.
- Automatically updates the Via of all outgoing transactions.
- Allows cancelling individual transactions or all outstanding transactions.

The **ISipTransactionStatefulProxySvc** is an ECOM object

The **ISipTransactionStatefulProxySvc** is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipTransactionStatefulProxySvc

Interface Id: IID_ISipTransactionStatefulProxySvc

A user can query the **ISipContext** (see page 23) to which this service is attached by calling **QueryIf** on it. It can also directly access all other services attached to the **ISipContext** (see page 23) through the same mean.

Location

SipProxy/ISipTransactionStatefulProxySvc.h

See Also

ISipTransactionStatefulProxyMgr (see page 428)

Methods

Method	Description
•   AddLocalRecordRoute (see page 437)	Creates a Record-Route header and adds it to the request. The host and port are later overridden based on the actual network interface and port used to send the packet.
•   AddRecordRoute (see page 437)	Creates a Record-Route header and adds it to the request.
•   CancelAllClientTransactions (see page 438)	Sends a CANCEL on all non-terminated client transactions.
•   ForwardRequest (see page 438)	Forwards a request.
•   ForwardResponse (see page 439)	Sends or forwards a response on the server transaction.
•   SetConfig (see page 439)	Sets the configuration object used by the stateful service.
•   SetManager (see page 440)	Configures the manager that receives events from this service.

Legend

	Method
	abstract

3.5.1.7.1 - Methods

3.5.1.7.1.1 - ISipTransactionStatefulProxySvc::AddLocalRecordRoute Method New in 4.1.4

Creates a Record-Route header and adds it to the request. The host and port are later overriden based on the actual network interface and port used to send the packet.

C++

```
virtual mxt_result AddLocalRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rRecordRouteNameAddr	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and the username is overriden. It must be either a SIP or a SIPS URI. The host/port portion is later overriden to reflect the actual network interface and port used to send the packet.
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

Returns

- resS_OK: The Record-Route has been added.
- resFE_INVALID_ARGUMENT: The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- resFE_FAIL: The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteNameAddr parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (it adds it if not present). It also creates a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity. Finally, it adds this Record-Route to the request in parameter.

This method differs from AddRecordRoute (see page 437) in that the host/port from rRecordRouteNameAddr is discarded and later replaced by the actual interface and port used to send the request.

See Also

ForwardRequest (see page 438), AddRecordRoute (see page 437)

3.5.1.7.1.2 - ISipTransactionStatefulProxySvc::AddRecordRoute Method

Creates a Record-Route header and adds it to the request.

C++

```
virtual mxt_result AddRecordRoute(INOUT CSipPacket& rRequest, IN const CNameAddr& rRecordRouteNameAddr, IN const CGenParamList* pRecordRouteParams) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rRecordRouteNameAddr	The URI to put in the Record-Route header. Note that the 'lr' parameter is added if not present and username is overriden. It must be either a SIP or a SIPS URI.
IN const CGenParamList* pRecordRouteParams	Optional header parameters to add to the Record-Route header.
rPacket	The request where the Record-Route is inserted.

Returns

- resS_OK: The Record-Route has been added.

- **resFE_INVALID_ARGUMENT:** The Record-Route could not be added either because the packet in parameter is not a request or the name-addr is not a SIP or SIPS URI.
- **resFE_FAIL:** The Record-Route could not be added.

Description

Creates a Record-Route header with the rRecordRouteUri in parameter and adds the optional pRecordRouteParams if present. This method also makes sure that the 'lr' parameter is present in the URI (that is it adds it if not present). It will also create a username that will be recognizable by this proxy later if seen in the context of this dialog. This username is computed from the dialog identifiers as well as this proxy identity.

Finally, it adds this Record-Route to the request in parameter.

See Also

ForwardRequest (see page 438)

3.5.1.7.1.3 - ISipTransactionStatefulProxySvc::CancelAllClientTransactions Method

Sends a CANCEL on all non-terminated client transactions.

C++

```
virtual mxt_result CancelAllClientTransactions() = 0;
```

Returns

- **resS_OK:** Success.
- **resFE_INVALID_STATE:** The object has not been configured with its manager.

Description

Sends a CANCEL on all non-terminated client transactions.

3.5.1.7.1.4 - ISipTransactionStatefulProxySvc::ForwardRequest Method

Forwards a request.

C++

```
virtual mxt_result ForwardRequest(IN CSipPacket& rRequest, IN mxt_opaque opqApplication, OUT ISipClientTransaction*& rpTransaction, IN uint64_t uTimeoutMs = 0) = 0;
```

Parameters

Parameters	Description
IN CSipPacket& rRequest	The request to forward. The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454()) on this parameter, once for each reference the caller has to this packet. This packet is used and modified by the stack before actually forwarding it.
IN mxt_opaque opqApplication	Application provided opaque identifier for the client transaction created to forward the request. This is not used by the stack, but it is instead reported as-is in the various events concerning this new transaction.
OUT ISipClientTransaction*& rpTransaction	When this method returns success, this parameter holds the transaction interface that offers some more functionality related to the newly created client transaction.
IN uint64_t uTimeoutMs = 0	Optional. The time, in milliseconds, after the EvRequestForwardTimeout event is reported to the manager when no final response to that forwarded request has been received. When the uTimeoutMs is 0, the timer is not started, i.e., the EvRequestForwardTimeout event is not reported. The default uTimeoutMs value is 0.

Returns

- **resS_OK:** Success.

- **resFE_INVALID_STATE**: The object was not configured with its manager.
- **resFE_INVALID_ARGUMENT**: The packet is a response.
- **failure code**: General failure cases.

Description

This method is used by the manager to create a new client transaction and forward on it a request it has received through the **ISipTransactionStatefulProxyMgr::EvTargetFound** (see page 431) and **ISipTransactionStatefulProxyMgr::EvNoTargetFound** (see page 429) events.

As described in the **ISipTransactionStatefulProxyMgr** (see page 428) interface, the manager must properly update the SIP request before forwarding it. This includes updating the request-URI (if necessary) and adding Record-Route headers.

The stack will later use the **ISipTransactionStatefulProxyMgr** (see page 428) interface to report the event about the newly created client transaction. This specific transactionOpaque parameter is provided back to the manager when reporting events for this client transaction.

See Also

ISipTransactionStatefulProxyMgr::EvTargetFound (see page 431), **ISipTransactionStatefulProxyMgr::EvNoTargetFound** (see page 429) **ISipTransactionStatefulProxyMgr::EvRequestForwardTimeout** (see page 431)

3.5.1.7.1.5 - **ISipTransactionStatefulProxySvc::ForwardResponse** Method

Sends or forwards a response on the server transaction.

C++

```
virtual mxt_result ForwardResponse( IN CSipPacket& rResponse ) = 0;
```

Parameters

Parameters	Description
IN CSipPacket& rResponse	<p>The response to forward.</p> <p>The caller must immediately release all references it has to this packet after calling this function. This is done by calling CSipPacket::Release (see page 454)() on this parameter, once for each reference the caller has to this packet.</p> <p>This packet is used and modified by the stack before actually forwarding it.</p>

Returns

- Success or failure code.

Description

This method is used to send a response on the server transaction. The manager can forward a response from a client transaction or it can also create its own response from the original request to send.

When forwarding a 2xx response, this service automatically cancels all remaining client branches.

This method can be used to answer requests that are specifically for this device, like a REGISTER request for this device's domain.

See Also

ISipTransactionStatefulProxyMgr::EvTransactionProgress (see page 434),
ISipTransactionStatefulProxyMgr::EvTransactionSucceeded (see page 435), **ISipTransactionStatefulProxyMgr::EvTransactionFailed** (see page 433)

3.5.1.7.1.6 - **ISipTransactionStatefulProxySvc::SetConfig** Method

Sets the configuration object used by the stateful service.

C++

```
virtual void SetConfig(IN const CSipProxyConfig& rConfig) = 0;
```

Parameters

Parameters	Description
IN const CSipProxyConfig& rConfig	The proxy configuration object.

Description

Sets the configuration object for this proxy service. It is recommended to call this method only once at service creation time to avoid potential problems when the configuration changes at runtime.

3.5.1.7.1.7 - ISipTransactionStatefulProxySvc::SetManager Method

Configures the manager that receives events from this service.

C++

```
virtual mxt_result SetManager(IN ISipTransactionStatefulProxyMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionStatefulProxyMgr* pMgr	Pointer to the manager. It cannot be NULL.

Returns

- resS_OK: Manager configured successfully.
- resFE_INVALID_ARGUMENT: NULL pointer provided as parameter.

Description

Configures the manager that receives events from this service.

3.6 - SipTransport

This section documents the Sources/SipTransport folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 440)

3.6.1 - Classes

This section documents the classes of the Sources/SipTransport folder.

Classes

Class	Description
CInstanceTracker (see page 440)	
CSipPacket (see page 447)	
ISipDataLogger (see page 458)	
ISipTlsContextFactory (see page 460)	
ISipTransportObserver (see page 467)	
ISipTransportUser (see page 470)	

3.6.1.1 - CInstanceTracker Class**Class Hierarchy**

```
CInstanceTracker
```

C++

```
class CInstanceTracker;
```

Description

This class gives static access to all watched instances. Using this class it is possible to get information about still allocated instances to help debug memory leaks. The following information can be accessed using this class:

- The number of still allocated watched instances.
- A vector containing pointers to still allocated watched instances
- A CBlob object containing information about still allocated watched instances.

Location

Tools/CInstanceTracker.h

Methods

Method	Description
◆ DebugGetContextDump (see page 443)	Gets a CBlob containing information about the ISipContext (see page 23) still allocated.
◆ DebugGetContextTable (see page 443)	Gets the vector of ISipContext (see page 23) still allocated.
◆ DebugGetNumContextsLeft (see page 443)	Gets the number of ISipContext (see page 23) still allocated.
◆ DebugGetNumPacketsLeft (see page 444)	Gets the number of CSipPacket (see page 447) still allocated.
◆ DebugGetNumRequestContextsLeft (see page 444)	Gets the number of ISipRequestContext still allocated.
◆ DebugGetNumTransactionsLeft (see page 444)	Gets the number of CSipTransaction still allocated.
◆ DebugGetPacketDump (see page 444)	Gets a CBlob containing information about the CSipPacket (see page 447) still allocated.
◆ DebugGetPacketTable (see page 445)	Gets the vector of CSipPacket (see page 447) still allocated.
◆ DebugGetRequestContextDump (see page 445)	Gets a CBlob containing information about the ISipRequestContext still allocated.
◆ DebugGetRequestContextTable (see page 446)	Gets the vector of ISipRequestContext still allocated.
◆ DebugGetTransactionDump (see page 446)	Gets a CBlob containing information about the CSipTransaction still allocated.
◆ DebugGetTransactionTable (see page 446)	Gets the vector of CSipTransaction still allocated.

Legend

◆	Method
---	--------

Structs

Struct	Description
SMemoryRecords (see page 441)	

3.6.1.1.1 - Structs

3.6.1.1.1.1 - CInstanceTracker::SMemoryRecords Struct

Class Hierarchy

C++

```
struct SMemoryRecords : public CMemoryAllocator::IMemoryBlockAccumulator {
    size_t m_uNumberOfAllocatedMemoryBlocks;
    const char* m_szTypeName;
    CMemoryAllocator::CMemoryBlock* m_astMemoryBlocks[uINSTANCE_TRACKER_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS];
};
```

Description

This structure is used to report some information about the instances of allocated memory blocks.

Constructors

Constructor	Description
◆ SMemoryRecords (see page 442)	Default Constructor.

Legend

	Method
--	--------

Destructors

Destructor	Description
 ~SMemoryRecords (see page 442)	Virtual destructor to avoid the warning caused by having another virtual function.

Legend

	Method
	virtual

3.6.1.1.1.1.1 - Data Members**3.6.1.1.1.1.1.1 - CInstanceTracker::SMemoryRecords::m_astMemoryBlocks Data Member**

Array of up to MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS of currently allocated memory blocks.

C++

```
CMemoryAllocator::CMemoryBlock* m_astMemoryBlocks[uINSTANCE_TRACKER_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS];
```

Description

If there are less than MXD_FRAMEWORK_FINALIZE_INFO_NUMBER_OF_STORED_LEAKED_MEMORY_BLOCKS allocated blocks, the m_uNumberOfAllocatedMemoryBlocks (see page 442) describes the number of blocks in the array.

3.6.1.1.1.1.1.2 - CInstanceTracker::SMemoryRecords::m_szTypeName Data Member

Memory block type which must be accumulated.

C++

```
const char* m_szTypeName;
```

3.6.1.1.1.1.1.3 - CInstanceTracker::SMemoryRecords::m_uNumberOfAllocatedMemoryBlocks Data Member

Count of total allocated memory blocks.

C++

```
size_t m_uNumberOfAllocatedMemoryBlocks;
```

3.6.1.1.1.1.2 - Constructors**3.6.1.1.1.1.2.1 - CInstanceTracker::SMemoryRecords::SMemoryRecords Constructor**

Default Constructor.

C++

```
SMemoryRecords();
```

Description

Default constructor for the SMemoryRecords that simply initializes structure values to their proper initial values.

3.6.1.1.1.1.3 - Destructors**3.6.1.1.1.1.3.1 - CInstanceTracker::SMemoryRecords::~SMemoryRecords Destructor**

Virtual destructor to avoid the warning caused by having another virtual function.

C++

```
virtual ~SMemoryRecords();
```

3.6.1.1.2 - Methods**3.6.1.1.2.1 - CInstanceTracker::DebugGetContextDump Method**

Gets a CBlob containing information about the ISipContext (see page 23) still allocated.

C++

```
static void DebugGetContextDump(OUT CBlob& rblobContextDump);
```

Parameters

Parameters	Description
OUT CBlob& rblobContextDump	The blob that contains information about all allocated contexts.

Description

Dumps into a blob information about all the contexts still allocated.

Notes

This method MUST be called only after the Shutdown is completed.

Example

This shows how to get a blob containing ISipContext (see page 23) information and print it.

```
CBlob blobDump;
CInstanceTracker::DebugGetContextDump(blobDump);
printf(reinterpret_cast<char*>(blobDump.GetFirstIndexPtr()));
```

3.6.1.1.2.2 - CInstanceTracker::DebugGetContextTable Method

Gets the vector of ISipContext (see page 23) still allocated.

C++

```
static uint16_t DebugGetContextTable(OUT CVector<ISipContext*>& rvecpContexts);
```

Parameters

Parameters	Description
OUT CVector<ISipContext*>& rvecpContexts	The vector to be filled with the ISipContext (see page 23) pointers that are allocated.

Returns

The number of allocated ISipContext (see page 23) instances.

Description

Gives access to a vector of ISipContext (see page 23) pointers that are allocated. This function will increase the reference count of the ISipContexts in the vector. It is the responsibility of the caller of this method to call ReleaselfRef for each ISipContext (see page 23) in the vector. To avoid race conditions this method MUST only be used in a debug scenario, and MUST be called from a thread using the same context as the SipCore thread after the stack has been shut down. Its main goal is to identify memory leaks.

Example

This shows how to have access to the vector of allocated ISipContext (see page 23) pointers.

```
CVector<ISipContext*> vecpWatchedSipContexts;
CInstanceTracker::DebugGetContextTable(OUT vecpWatchedSipContexts);
```

3.6.1.1.2.3 - CInstanceTracker::DebugGetNumContextsLeft Method

Gets the number of ISipContext (see page 23) still allocated.

C++

```
static unsigned int DebugGetNumContextsLeft();
```

Returns

The number of allocated ISipContext (see page 23).

Description

Will return the number of allocated ISipContext (see page 23).

3.6.1.1.2.4 - CInstanceTracker::DebugGetNumPacketsLeft Method

Gets the number of CSipPacket (see page 447) still allocated.

C++

```
static unsigned int DebugGetNumPacketsLeft();
```

Returns

The number of allocated CSipPacket (see page 447).

Description

Returns the number of allocated CSipPacket (see page 447).

3.6.1.1.2.5 - CInstanceTracker::DebugGetNumRequestContextsLeft Method

Gets the number of ISipRequestContext still allocated.

C++

```
static unsigned int DebugGetNumRequestContextsLeft();
```

Returns

The number of allocated ISipRequestContext.

Description

Returns the number of allocated ISipRequestContext.

3.6.1.1.2.6 - CInstanceTracker::DebugGetNumTransactionsLeft Method

Gets the number of CSipTransaction still allocated.

C++

```
static unsigned int DebugGetNumTransactionsLeft();
```

Returns

The number of allocated CSipTransaction.

Description

Returns the number of allocated CSipTransaction.

3.6.1.1.2.7 - CInstanceTracker::DebugGetPacketDump Method

Gets a CBlob containing information about the CSipPacket (see page 447) still allocated.

C++

```
static void DebugGetPacketDump(OUT CBlob& rblobPacketDump);
```

Parameters

Parameters	Description
OUT CBlob& rblobPacketDump	The blob that contains information about all allocated packets.

Description

Dumps into a blob information about all the packets still allocated.

Notes

This method MUST be called only after the Shutdown is completed.

Example

This shows how to get a blob containing CSipPacket (see page 447) information and print it.

```
CBlob blobDump;
CInstanceTracker::DebugGetPacketDump(blobDump);
printf(reinterpret_cast<char*>(blobDump.GetFirstIndexPtr()));
```

3.6.1.1.2.8 - CInstanceTracker::DebugGetPacketTable Method

Gets the vector of CSipPacket (see page 447) still allocated.

C++

```
static unsigned int DebugGetPacketTable(OUT CVector<CSipPacket*>& rvecpPacket);
```

Parameters

Parameters	Description
rvecpPackets	The vector to be filled with the CSipPacket (see page 447) pointers that are allocated.

Returns

The number of allocated CSipPacket (see page 447) instances.

Description

Gives access to a vector of CSipPacket (see page 447) pointers that are allocated. This function will increase the reference count of the CSipPacket (see page 447) in the vector. It is the responsibility of the caller of this method to call Release for each CSipPacket (see page 447) in the vector. To avoid race conditions this method MUST only be used in a debug scenario, and MUST be called after the stack has been shut down. Its main goal is to identify memory leaks.

Example

This shows how to have access to the vector of allocated CSipPacket (see page 447) pointers.

```
CVector<CSipPacket*> vecpWatchedSipPackets;
CInstanceTracker::DebugGetPacketTable(OUT vecpWatchedSipPackets);
```

3.6.1.1.2.9 - CInstanceTracker::DebugGetRequestContextDump Method

Gets a CBlob containing information about the ISipRequestContext still allocated.

C++

```
static void DebugGetRequestContextDump(OUT CBlob& rblobRequestContextDump);
```

Parameters

Parameters	Description
OUT CBlob& rblobRequestContextDump	The blob that contains information about all allocated request contexts.

Description

Dumps into a blob information about all the request contexts still allocated.

Notes

This method MUST be called only after the Shutdown is completed.

Example

This shows how to get a blob containing ISipRequestContext information and print it.

```
CBlob blobDump;
CInstanceTracker::DebugGetRequestContextDump(blobDump);
printf(reinterpret_cast<char*>(blobDump.GetFirstIndexPtr()));
```

3.6.1.1.2.10 - CInstanceTracker::DebugGetRequestContextTable Method

Gets the vector of ISipRequestContext still allocated.

C++

```
static unsigned int DebugGetRequestContextTable(OUT CVector<ISipRequestContext*>& rvecpRequestContext);
```

Parameters

Parameters	Description
rvecpRequestContexts	The vector to be filled with the ISipRequestContexts pointers that are allocated.

Description

Gives access to a vector of ISipRequestContext pointers that are allocated. This function will increase the reference count of the ISipRequestContexts in the vector. It is the responsibility of the caller of this method to call ReleaseSelfRef for each ISipRequestContexts in the vector. To avoid race conditions this method MUST only be used in a debug scenario, and MUST be called after the stack has been shut down. Its main goal is to identify memory leaks.

Example

This shows how to have access to the vector of allocated ISipRequestContext pointers.

```
CVector<ISipRequestContext*> vecpWatchedSipRequestContexts;
CInstanceTracker::DebugGetRequestContextTable(
    OUT vecpWatchedSipRequestContexts);
```

3.6.1.1.2.11 - CInstanceTracker::DebugGetTransactionDump Method

Gets a CBlob containing information about the CSipTransaction still allocated.

C++

```
static void DebugGetTransactionDump(OUT CBlob& rblobTransactionDump);
```

Parameters

Parameters	Description
OUT CBlob& rblobTransactionDump	The blob that contains information about all allocated transactions.

Description

Dumps into a blob information about all the transactions still allocated.

Notes

This method MUST be called only after the Shutdown is completed.

Example

This shows how to get a blob containing CSipTransaction information and print it.

```
CBlob blobDump;
CInstanceTracker::DebugGetTransactionDump(blobDump);
printf(reinterpret_cast<char*>(blobDump.GetFirstIndexPtr()));
```

3.6.1.1.2.12 - CInstanceTracker::DebugGetTransactionTable Method

Gets the vector of CSipTransaction still allocated.

C++

```
static unsigned int DebugGetTransactionTable(OUT CVector<CSipTransaction*>& rvecpTransaction);
```

Parameters

Parameters	Description
rvecpTransactions	The vector to be filled with the CSipTransaction pointers that are allocated.

Returns

The number of allocated CSipTransaction instances.

Description

Gives access to a vector of CSipTransaction pointers that are allocated. To avoid race conditions this method MUST only be used in a debug scenario, and MUST be called after the stack has been shut down. Its main goal is to identify memory leaks.

Example

This shows how to have access to the vector of allocated CSipTransaction pointers.

```
CVector<CSipTransaction*> vecpWatchedSipTransactions;
CInstanceTracker::DebugGetTransactionTable(
    OUT vecpWatchedSipTransactions);
```

3.6.1.2 - CSipPacket Class

Class Hierarchy



C++

```
class CSipPacket : public CSipPacketParser, public CWatchedInstance<CSipPacket>;
```

Description

This class is used as a container for a packet sent and received. It is derived from the CSipPacketParser (see page 312) class, which is used to parse a raw header, or on the contrary, to serialize a series of headers and a payload. The present class also adds other useful data such as the local and remote IP addresses and ports and the transport used.

The SIP stack follows a mechanism to process incoming and outgoing packets. When a packet is received from the network, only one owner can access it. Once the SipParser determines that it has received the whole packet, it becomes a const instance and it can be shared and reference counted throughout the SIP stack and possibly the application. The same mechanism is also true when sending a packet. There is only one owner that can access the packet until it is complete and ready to be sent. Once every services have updated the packet, it becomes const and can be shared throughout several threads.

Constant methods that can modify the packet headers are synchronized internally so different threads accessing them do not need to bother with synchronization.

The following is the general SIP stack usage of SIP packets:

- SIP packets are never shared or reference counted until they are complete.
- Only a complete SIP packet can be shared or reference counted.
- When shared, a SIP packet is always const.

Location

SipTransport/CSipPacket.h

Constructors

Constructor	Description
CSipPacket (see page 449)	Default Constructor.

CSipPacketParser Class

CSipPacketParser Class	Description
CSipPacketParser (see page 313)	Constructor.

Legend

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

Destructors**CSipPacketParser Class**

CSipPacketParser Class	Description
~CSipPacketParser (see page 314)	Destructor.

Legend

	Method
	virtual

Operators

Operator	Description
= (see page 457)	Operator.

CSipPacketParser Class

CSipPacketParser Class	Description
= (see page 321)	Assignment operator.

Legend

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

Methods

Method	Description
AddRef (see page 451)	Adds a reference to the object.
AddServerToResponses (see page 451)	Sets whether or not to automatically put the Server header in responses.
AddUserAgentToRequests (see page 452)	Sets whether or not to automatically put the User-Agent header in requests.
GetLocalAddr (see page 452)	Returns the local IP and port address.
GetMaxSize (see page 452)	Gets the maximum size of the serialized packet.
GetNextHopUri (see page 452)	Gets the next-hop URI.
GetNextHopUriHostname (see page 453)	Gets the next-hop URI hostname.
GetPeerAddr (see page 453)	Returns the peer IP and port address.
GetTransport (see page 453)	Returns the transport protocol ID.
IsNewConnectionAllowed (see page 453)	True if a new connection is allowed, false otherwise.
IsReceivedOnAuthenticatedConnection (see page 454)	Tells if the TLS connection associated with this packet authenticated the remote peer.
IsStatHandledForReception (see page 454)	Tells if the packet has been handled by the statistics container.
Release (see page 454)	Releases a reference to the object.
SetAllowNewConnection (see page 454)	Sets whether or not a new connection is allowed.
SetEntityId (see page 455)	Sets the identity of this entity to put in the User-Agent and the Server headers when generating requests and responses respectively.
SetLocalAddr (see page 455)	Sets the local IP and port address.
SetMaxForwards (see page 455)	Sets the value to put in the Max-Forwards header to put in the requests created by this class.
SetMaxSize (see page 456)	Sets the maximum size of the serialized packet.
SetNextHopUri (see page 456)	Sets the next-hop URI.
SetPeerAddr (see page 456)	Sets the peer IP and port address.
SetReceivedOnAuthenticatedConnection (see page 457)	Set the TLS authentication.
SetStatHandledForReception (see page 457)	Sets the value to tell if the packet has been handled in the statistics container.
SetTransport (see page 457)	Sets the transport protocol ID.

CSipPacketParser Class

CSipPacketParser Class	Description
AppendRawData (See page 314)	Buffers socket-received data. It also tries to parse the start line if not set. Input must be a NULL-terminated string.
CommitRawDataList (See page 315)	Instructs CSipPacketParser (See page 312) to use its list of buffered raw headers and build its CHeaderList (See page 155) with them.
CreateSipMessageBody (See page 315)	Creates the message-body, it can be NULL.
GetHeaderList (See page 316)	Provides access to the header list.
GetPayload (See page 316)	Provides access to the optional payload.
GetRawDataList (See page 316)	Provides access to the list of buffered headers.
GetRequestLine (See page 317)	Provides access to the request line if available.
GetSipMessageBody (See page 317)	Gets the message-body. Gets the message-body.
GetStatusLine (See page 318)	Provides access to the status line if available.
IsLocallyGenerated (See page 318)	Returns true if the packet is locally generated.
IsRequest (See page 318)	Returns true if the packet is a request.
IsResponse (See page 319)	Returns true if the packet is a response.
Reset (See page 319)	Clears all data back to its initial state.
Serialize (See page 319)	Outputs the packet's content into the buffer.
SetLocallyGenerated (See page 319)	Sets the local packet generation indicator.
SetPayload (See page 320)	Sets the optional payload.
SetRequestLine (See page 320)	Sets the request line. Converts this packet to a request.
SetSipMessageBody (See page 320)	Sets the message-body.
SetStatusLine (See page 320)	Sets the status line. Converts this packet to a response.

Legend

	Method
--	--------

Enumerations

Enumeration	Description
ERecordRouteProcessing (See page 458)	

Structs

CSipPacketParser Class

CSipPacketParser Class	Description
SRawData (See page 313)	

3.6.1.2.1 - Constructors

3.6.1.2.1.1 - CSipPacket

3.6.1.2.1.1.1 - CSipPacket::CSipPacket Constructor

Default Constructor.

C++

```
CSipPacket();
```

Description

Creates a new packet. Since it is a new object (instance), the reference count is set to 1. By default, a new connection is allowed, addresses and transport id are invalid.

3.6.1.2.1.1.2 - CSipPacket::CSipPacket Constructor

Copy Constructor.

C++

```
CSipPacket(IN const CSipPacket& rSrc);
```

Parameters

Parameters	Description
IN const CSipPacket& rSrc	The source packet copied into the new packet.

Description

Copy constructor.

Creates a new packet from another one.

Since it is a new object (instance) and not a reference to the source object (instance), the reference count is set to 1.

By default, a new connection is allowed.

3.6.1.2.1.1.3 - CSipPacket::CSipPacket Constructor

Constructor for outgoing responses.

C++

```
CSipPacket(IN const CSipPacket& rRequest, IN unsigned int uCode, IN const char* szReason, IN
ERecordRouteProcessing eRrProcessing, IN TO CHeaderList* pExtraHeaders = NULL, IN TO CSipMessageBody*
pMessageBody = NULL, IN bool bSetViaRawHeaderToNull = true);
```

Parameters

Parameters	Description
IN const CSipPacket& rRequest	The request to which this response will respond.
IN unsigned int uCode	The status code for that response.
IN const char* szReason	The reason phrase for that response.
IN ERecordRouteProcessing eRrProcessing	When eCopyRecordRoute, copy Record-Route headers from the request to the response. When eDoNotCopyRecordRoute, the Record-Route headers are not copied in the response. Record-Route headers MUST be copied in responses that establish a dialog.
IN TO CHeaderList* pExtraHeaders = NULL	Extra headers to add to this request. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody = NULL	Message-body information to add to this request. It can be NULL. Ownership is TAKEN.
IN bool bSetViaRawHeaderToNull = true	True if the raw header of the Via header must be set to NULL (thus forcing the serialization of the Via Header). False if the raw header of the Via header must not be set to NULL. False is passed by default.

Description

Extended constructor. Creates a response from the specified request. It adds RFC mandatory headers in the response: From, To, Call-ID, CSeq, and Via headers. It adds a To-tag when none is present in the request and the response is different from 100 Trying. It copies the Timestamp header from the request to the response if the response code is 100 Trying. It adds the headers in the extra headers and message body parameters.

It adds a Server header created with the value configured through the SetServer method, unless a Server header is already present in the extra headers.

It adds the proper Content-Length header according to the message body.

See Also

SetServer

3.6.1.2.1.1.4 - CSipPacket::CSipPacket Constructor

Constructor for outgoing requests.

C++

```
CSipPacket(IN const CString& rstrMethod, IN TO CHeaderList* pExtraHeaders = NULL, IN TO CSipMessageBody*
pMessageBody = NULL);
```

Parameters

Parameters	Description
IN const CString& rstrMethod	The method for this request.
IN TO CHeaderList* pExtraHeaders = NULL	Extra headers to add to this request. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody = NULL	Message-body information to add to this request. It can be NULL. Ownership is TAKEN.

Description

Extended constructor. This method creates a basic request. It creates a request line and sets its method with the method in parameter.

In addition to the headers in the extra headers and content info parameters, it adds a User-Agent header with the value configured through the SetUserAgent method, unless a User-Agent header is already present in the extra headers.

Note that it also adds the proper Content-Length header according to the content info.

See Also

[SetUserAgent](#)

3.6.1.2.2 - Methods

3.6.1.2.2.1 - CSipPacket::AddRef Method

Adds a reference to the object.

C++

```
unsigned int AddRef() const;
```

Returns

The CSipPacket ([see page 447](#))'s reference count after being increased.

Description

This method is used to increment the reference count number on a CSipPacket ([see page 447](#)) instance. A reference is required when any object wants to keep a pointer to the CSipPacket ([see page 447](#)) after passing the CSipPacket ([see page 447](#)) to another object, to ensure that the CSipPacket ([see page 447](#)) object is not released from memory by another object.

See Also

[Release](#) ([see page 454](#))

3.6.1.2.2.2 - CSipPacket::AddServerToResponses Method

Sets whether or not to automatically put the Server header in responses.

C++

```
static void AddServerToResponses(IN bool bAdd);
```

Parameters

Parameters	Description
IN bool bAdd	True if the Server header should be added to responses, false otherwise.

Description

Sets whether or not the Server header should be added to responses. If it is added, its value is the one configured through the SetEntityId ([see page 455](#)) method. Note that it is added **only** when the response constructor (the constructor that takes a request as its first parameter, a status code as its second, and a reason phrase as its third) is used.

See Also

[SetEntityId](#) ([see page 455](#))

3.6.1.2.2.3 - CSipPacket::AddUserAgentToRequests Method

Sets whether or not to automatically put the User-Agent header in requests.

C++

```
static void AddUserAgentToRequests( IN bool bAdd );
```

Parameters

Parameters	Description
IN bool bAdd	True if the User-Agent header should be added to requests, false otherwise.

Description

Sets whether or not the User-Agent header should be added to requests. If it is added, its value is the one configured through the SetEntityId (see page 455) method. Note that it is added **only** when the request constructor (the constructor that takes a method as its first parameter) is used.

See Also

SetEntityId (see page 455)

3.6.1.2.2.4 - CSipPacket::GetLocalAddr Method

Returns the local IP and port address.

C++

```
const CSocketAddr& GetLocalAddr() const;
```

Returns

The local address.

Description

Returns the local address.

3.6.1.2.2.5 - CSipPacket::GetMaxSize Method

Gets the maximum size of the serialized packet.

C++

```
unsigned int GetMaxSize() const;
```

Returns

The maximum size of the packet.

Description

Gets the maximum serialized size of the packet. Note that this maximum size is not enforced by this class and should instead be enforced by users. By default, the maximum size of a packet is very large (i.e. infinite for all practical purposes).

3.6.1.2.2.6 - CSipPacket::GetNextHopUri Method

Gets the next-hop URI.

C++

```
const CSipUri& GetNextHopUri() const;
```

Returns

The next-hop URI.

Description

Gets the next-hop URI.

See Also

[SetNextHopUri](#) (see page 456)

3.6.1.2.2.7 - CSipPacket::GetNextHopUriHostname Method

Gets the next-hop URI hostname.

C++

```
const CString& GetNextHopUriHostname() const;
```

Returns

The hostname.

Description

Gets the next-hop URI hostname.

See Also

[GetNextHopUri](#) (see page 452)

3.6.1.2.2.8 - CSipPacket::GetPeerAddr Method

Returns the peer IP and port address.

C++

```
const CSocketAddr& GetPeerAddr() const;
```

Returns

The peer address.

Description

Returns the peer address.

3.6.1.2.2.9 - CSipPacket::GetTransport Method

Returns the transport protocol ID.

C++

```
ESipTransport GetTransport() const;
```

Returns

The transport protocol ID.

Description

Returns the transport protocol ID.

3.6.1.2.2.10 - CSipPacket::IsNewConnectionAllowed Method

True if a new connection is allowed, false otherwise.

C++

```
bool IsNewConnectionAllowed() const;
```

Returns

True if new connection allowed, false otherwise.

Description

Returns whether or not a new connection is allowed.

3.6.1.2.2.11 - CSipPacket::IsReceivedOnAuthenticatedConnection Method

Tells if the TLS connection associated with this packet authenticated the remote peer.

C++

```
bool IsReceivedOnAuthenticatedConnection() const;
```

Returns

The TLS authentication value.

Description

Tells whether or not the remote peer has been authenticated when creating the TLS connection used to receive this packet.

See Also

[SetReceivedOnAuthenticatedConnection](#) (see page 457)

3.6.1.2.2.12 - CSipPacket::IsStatHandledForReception Method

Tells if the packet has been handled by the statistics container.

C++

```
bool IsStatHandledForReception() const;
```

Returns

True if the packet has been handled by the statistics container.

Description

Tells whether or not the packet has been handled by the statistics container.

3.6.1.2.2.13 - CSipPacket::Release Method

Releases a reference to the object.

C++

```
unsigned int Release() const;
```

Returns

The CSipPacket (see page 447)'s reference count after being decreased.

Description

This method is used to decrement the CSipPacket (see page 447)'s reference count. This needs to be called when the object owning a reference to the CSipPacket (see page 447) no longer has any use for this CSipPacket (see page 447). When the reference count reaches zero, the CSipPacket (see page 447) deletes itself.

See Also

[AddRef](#) (see page 451)

3.6.1.2.2.14 - CSipPacket::SetAllowNewConnection Method

Sets whether or not a new connection is allowed.

C++

```
void SetAllowNewConnection(IN bool bAllow);
```

Parameters

Parameters	Description
IN bool bAllow	The value to be stored.

Description

Sets whether or not a new connection is allowed.

3.6.1.2.2.15 - CSipPacket::SetEntityId Method

Sets the identity of this entity to put in the User-Agent and the Server headers when generating requests and responses respectively.

C++

```
static void SetEntityId(IN const CString& rstrUserId, IN const CString& rstrServerId);
```

Parameters

Parameters	Description
IN const CString& rstrServerId	The string to put in the Server header generated by this class.
rstrAppId	The string to put in the User-Agent header generated by this class.

Description

Sets the string to put in the User-Agent header and the Server header of requests and responses generated by this class. This string should follow the ABNF for User-Agent and Server defined in RFC 3261.

rstrId	= server-val *(LWS server-val)
server-val	= product / comment
product	= token [SLASH product-version]
product-version	= token
comment	= LPAREN *(ctext / quoted-pair / comment) RPAREN

Note however that this class makes no verification on the validity of this value.

See Also

AddUserAgentToRequests (see page 452), AddServerToResponses (see page 451)

3.6.1.2.2.16 - CSipPacket::SetLocalAddr Method

Sets the local IP and port address.

C++

```
void SetLocalAddr(IN const CSocketAddr& rAddr);
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddr	The local address to be stored.

Description

Sets the local address.

3.6.1.2.2.17 - CSipPacket::SetMaxForwards Method

Sets the value to put in the Max-Forwards header to put in the requests created by this class.

C++

```
static void SetMaxForwards(IN unsigned int uMaxForwards);
```

Parameters

Parameters	Description
IN unsigned int uMaxForwards	The value to put in the Max-Forwards header of the request created by the CSipPacket (see page 447) class.

Description

Sets the value to put in the Max-Forwards header of the request created by the CSipPacket (see page 447) class. By default, this value is 70, the value recommended by RFC 3261.

3.6.1.2.2.18 - CSipPacket::SetMaxSize Method

Sets the maximum size of the serialized packet.

C++

```
void SetMaxSize(IN unsigned int uMaxSize);
```

Parameters

Parameters	Description
IN unsigned int uMaxSize	The value to be stored.

Description

Sets the maximum serialized size of the packet. Note that this maximum size is not enforced by this class and should instead be enforced by users.

3.6.1.2.2.19 - CSipPacket::SetNextHopUri Method

Sets the next-hop URI.

C++

```
void SetNextHopUri(IN const CSipUri& rNextHopUri);
```

Parameters

Parameters	Description
IN const CSipUri& rNextHopUri	The next-hop URI.

Description

Sets the next-hop URI.

See Also

GetNextHopUri (see page 452)

3.6.1.2.2.20 - CSipPacket::SetPeerAddr Method

Sets the peer IP and port address.

C++

```
void SetPeerAddr(IN const CSocketAddr& rAddr);
```

Parameters

Parameters	Description
IN const CSocketAddr& rAddr	The peer address to be stored.

Description

Sets the peer address.

3.6.1.2.2.21 - CSipPacket::SetReceivedOnAuthenticatedConnection Method

Set the TLS authentication.

C++

```
void SetReceivedOnAuthenticatedConnection(IN bool bPeerAuth);
```

Parameters

Parameters	Description
bAuth	Boolean telling whether or not TLS authentication has been used.

Description

Sets the TLS peer authentication. True if TLS authentication has been used on the socket that received this packet, otherwise false.

See Also

IsReceivedOnAuthenticatedConnection (see page 454)

3.6.1.2.2.22 - CSipPacket::SetStatHandledForReception Method

Sets the value to tell if the packet has been handled in the statistics container.

C++

```
void SetStatHandledForReception(IN bool bHandled) const;
```

Parameters

Parameters	Description
IN bool bHandled	True if the packet has been handled by the statistics container.

Description

Sets whether or not the packet has been handled by the statistics container.

3.6.1.2.2.23 - CSipPacket::SetTransport Method

Sets the transport protocol ID.

C++

```
void SetTransport(IN ESipTransport transport);
```

Parameters

Parameters	Description
IN ESipTransport transport	The transport protocol ID to be stored.

Description

Sets the transport protocol ID.

3.6.1.2.3 - Operators

3.6.1.2.3.1 - CSipPacket::= Operator

Operator.

C++

```
CSipPacket& operator =(IN const CSipPacket& rSrc);
```

Parameters

Parameters	Description
IN const CSipPacket& rSrc	The source object copied to this one.

Description

Copies the object passed in parameter into the current object. However, since it keeps no reference whatsoever to the source object (instance) and has its own history, its original reference count remains unchanged.

3.6.1.2.4 - Enumerations

3.6.1.2.4.1 - CSipPacket::ERecordRouteProcessing Enumeration

```
enum ERecordRouteProcessing {
    eCopyRecordRoute,
    eDoNotCopyRecordRoute
};
```

Description

Indicate whether or not the Record route must be copied.

Members

Members	Description
eCopyRecordRoute	Use this value when the response can establish a dialog.
eDoNotCopyRecordRoute	Use this value when the response cannot establish a dialog.

3.6.1.3 - ISipDataLogger Class

Class Hierarchy

ISipDataLogger

C++

```
class ISipDataLogger;
```

Description

This is the interface that allows access to the SIP data that is being sent to the network and received from the network. This data is usually used for logging incoming and outgoing SIP traffic.

The SIP data is provided by the stack in two different formats.

Raw Data:

Raw data is provided through the LogRawData (see page 459) API of this interface. This API is called by the stack whenever data is actually sent on the network or received from the network. The data is provided as a buffer of bytes through a CBlob instance.

Users implementing this API must be aware that it is possible the raw data can potentially contain non-printable characters. The payload SIP can transport is not limited to text, it can very well be binary.

SIP Packet:

The CSipPacket (see page 447) is provided through the LogSipPacket (see page 460) API of this interface. This API is called by the stack on the following occasions:

- Upon the parsing of a complete and valid SIP packet and its payload. A valid SIP packet has valid From, To, Call-Id, CSeq, and top Via headers. In TCP, the packet must also have a valid Content-Length

header. The SIP packet must not be bigger than the configured maximal packet size, and then same holds true for the payload. See `ISipCoreConfig::SetMaxReceivePacketSize` (see page 53) and `ISipCoreConfig::SetMaxPayloadSize` (see page 52).

- When a SIP packet is associated with a new or existing connection and

it is being prepared to be sent on the network. Note that the packet is traced **before** actually being sent. It is still possible that the packet is not sent on the network, or it may be delayed before it is sent.

One cannot assume that no packets were received when `LogSipPacket` (see page 460) is not called, as packets with some level of syntax error are not logged through this. Moreover, when `LogSipPacket` (see page 460) is called for an outgoing SIP packet, one cannot assume that the packet is or will actually be sent on the network. `LogRawData` (see page 459) should be used to have this level of precision.

Location

`SipTransport/ISipDataLogger.h`

Destructors

Destructor	Description
~ISipDataLogger (see page 459)	Destructor.

Legend

	Method
	virtual

Methods

Method	Description
 <code>LogRawData</code> (see page 459)	Reports to the user that data has been received or sent.
 <code>LogSipPacket</code> (see page 460)	Reports to the user that a SIP packet has been received or sent.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
<code>EDirection</code> (see page 460)	This dictates the direction of the packet upon being logged.

3.6.1.3.1 - Destructors

3.6.1.3.1.1 - `ISipDataLogger::~ISipDataLogger` Destructor

Destructor.

C++

```
virtual ~ISipDataLogger();
```

3.6.1.3.2 - Methods

3.6.1.3.2.1 - `ISipDataLogger::LogRawData` Method

Reports to the user that data has been received or sent.

C++

```
virtual void LogRawData(IN EDirection eDir, IN const CSocketAddr& rLocalAddr, IN const CSocketAddr& rPeerAddr,  
IN const CBlob& rRawData, IN unsigned int uSize) = 0;
```

Parameters

Parameters	Description
IN EDirection eDir	The direction of the packet being logged.
IN const CSocketAddr& rLocalAddr	The local address associated with the raw data.
IN const CSocketAddr& rPeerAddr	The peer address associated with the raw data.
IN const CBlob& rRawData	The raw data coming from or going on the network.
IN unsigned int uSize	The size of the raw data.

Description

This method is called to let the implementing class log the raw data coming from or going on the network. In the case of data transmission, this method is called when the transmission is successful. It can be incomplete and then uSize is set accordingly.

3.6.1.3.2.2 - ISipDataLogger::LogSipPacket Method

Reports to the user that a SIP packet has been received or sent.

C++

```
virtual void LogSipPacket(IN EDirection eDir, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN EDirection eDir	The direction of the packet being logged.
IN const CSipPacket& rPacket	The packet to be logged.

Description

This method is called to let the implementing class log the packet. Note that when receiving a SIP packet, this is called only for complete and well-formed packets. See class description for more information.

3.6.1.3.3 - Enumerations

3.6.1.3.3.1 - ISipDataLogger::EDirection Enumeration

This dictates the direction of the packet upon being logged.

C++

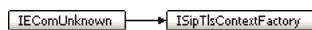
```
enum EDirection {
    eRECEIVED,
    eSENT,
    eLOOPBACK
};
```

Members

Members	Description
eRECEIVED	Received raw data or packet.
eSENT	Sent raw data or packet.
eLOOPBACK	Packet sent in loopback.

3.6.1.4 - ISipTlsContextFactory Class New in 4.1.4

Class Hierarchy



C++

```
class ISipTlsContextFactory : public IEComUnknown;
```

Description

This class represents a factory for the client and server TLS contexts. It keeps a default client and a default server context. These default contexts are used by the SIP stack when there are no contexts that map to a host name or an IP address. This class also keeps a list of client contexts grouped by peer host name and a list of server contexts grouped by local listening addresses.

The following are the responsibilities of the application when configuring TLS contexts:

- Set a default client TLS context with authentication enabled, the trusted certificates, and possibly the chain containing a personal certificate. The authentication must be enabled on client connections because this is the only way to identify the remote peer. The personal certificate is not mandatory for User-Agents as they usually do not possess such certificate.
- Could also optionally set a client TLS context per remote peer host name.
- Set a default server TLS context with, most of time, no authentication and a personal certificate in the chain.
- Set a list of server TLS contexts associated with a local listening address and asking for mutual authentication. When an application wants to do mutual authentication on TLS server connections, it needs to set a specific context per local listening address. In order for a server socket to authenticate a remote peer, the server **MUST** know the list of clients it is allowed to accept connections from. This is done by specifying a list of Subject Alternate Names that are to be accepted. Note that when the TLS peer authentication is not used on incoming server connections, the application **SHOULD** challenge the request by using the Digest Server Authentication to identify the peer.

The class `CTlsContext` allows to set all information required by client and server contexts.

The `ISipTlsContextFactory` is an ECOM interface obtained through the `ISipCoreConfig::GetTlsContextFactory` (see page 40) method. Hence, it is reference counted.

Location

`SipTransport/ISipTlsContextFactory.h`

See Also

`CTlsContext`, `ISipCoreConfig` (see page 28)

Methods

Method	Description
•  <code>AddTlsClientContextS</code> (see page 462)	Adds a TLS client context identified by the remote peer host name.
•  <code>AddTlsServerContextS</code> (see page 462)	Adds a TLS server context identified by local listening address.
•  <code>GetDefaultTlsClientContextS</code> (see page 462)	Gets the default TLS client context.
•  <code>GetDefaultTlsServerContextS</code> (see page 463)	Gets the default TLS server context.
•  <code>GetTlsClientContextS</code> (see page 463)	Gets the TLS client context corresponding to the remote peer host name.
•  <code>GetTlsServerContextS</code> (see page 464)	Gets the TLS server context corresponding to the local listening address.
•  <code>RemoveTlsClientContextS</code> (see page 464)	Removes the TLS client context identified by the remote peer host name.
•  <code>RemoveTlsServerContextS</code> (see page 465)	Removes the TLS server context identified by the remote peer socket address.
•  <code>SetDefaultTlsClientContextS</code> (see page 465)	Sets the default TLS client context.
•  <code>SetDefaultTlsServerContextS</code> (see page 465)	Sets the default TLS server context.
•  <code>UpdateTlsClientContextS</code> (see page 466)	Updates the TLS client context identified by the remote peer host name.
•  <code>UpdateTlsServerContextS</code> (see page 466)	Updates the TLS server context identified by the remote peer socket address.

Legend

	Method
	abstract

3.6.1.4.1 - Methods

3.6.1.4.1.1 - **ISipTlsContextFactory::AddTlsClientContextS** Method

Adds a TLS client context identified by the remote peer host name.

C++

```
virtual mxt_result AddTlsClientContextS(IN const CString& rstrHostname, IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrHostname	The remote peer host name.
IN const CTlsContext& rTlsContext	The TLS client context.

Returns

- resS_OK: The TLS client context is properly added.
- resFE_INVALID_STATE: A TLS client context already exists for the specified host name. Use [UpdateTlsClientContextS](#) (see page 466) to modify the TLS client context for that host name.
- resFE_FAIL: Unable to process message.

Description

Adds a TLS client context in the map with the remote peer host name as the key. If there are existing client TLS connections to the specified host name, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

[UpdateTlsClientContextS](#) (see page 466), [RemoveTlsClientContextS](#) (see page 464), [GetTlsClientContextS](#) (see page 463).

3.6.1.4.1.2 - **ISipTlsContextFactory::AddTlsServerContextS** Method

Adds a TLS server context identified by local listening address.

C++

```
virtual mxt_result AddTlsServerContextS(IN const CSocketAddr& rLocalAddr, IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rLocalAddr	The local listening address.
IN const CTlsContext& rTlsContext	The TLS server context.

Returns

- resS_OK: The TLS server context is properly added.
- resFE_INVALID_STATE: A TLS server context already exists for the specified local listening address. Use [UpdateTlsClientContextS](#) (see page 466) to modify the TLS server context for that local listening address.
- resFE_FAIL: Unable to process message.

Description

Adds a TLS server context in the map with the local listening address as the key. If there are existing accepted TLS connections using the specified local listening address, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

[UpdateTlsServerContextS](#) (see page 466), [RemoveTlsServerContextS](#) (see page 465), [GetTlsServerContextS](#) (see page 464).

3.6.1.4.1.3 - **ISipTlsContextFactory::GetDefaultTlsClientContextS** Method

Gets the default TLS client context.

C++

```
virtual mxt_result GetDefaultTlsClientContextS(OUT CTlsContext& rTlsContext) const = 0;
```

Parameters

Parameters	Description
OUT CTlsContext& rTlsContext	The default TLS client context.

Returns

- resS_OK: The default TLS client context is returned in the rTlsContext parameter.
- resFE_FAIL: Unable to process the message.

Description

Gets the default TLS client context set by the application. This is the generic context that applies to an outgoing TLS connection request when there is no specific context for a remote peer host name.

See Also

[SetDefaultTlsClientContextS](#) (see page 465)

3.6.1.4.1.4 - ISipTlsContextFactory::GetDefaultTlsServerContextS Method

Gets the default TLS server context.

C++

```
virtual mxt_result GetDefaultTlsServerContextS(OUT CTlsContext& rTlsContext) const = 0;
```

Parameters

Parameters	Description
OUT CTlsContext& rTlsContext	The default TLS server context.

Returns

- resS_OK: The default TLS server context is returned in the rTlsContext parameter.
- resFE_FAIL: Unable to process the message.

Description

Gets the default TLS server context set by the application. This is the generic context that applies to an incoming TLS connection request when there is no specific context for the local listening address.

See Also

[SetDefaultTlsServerContextS](#) (see page 465)

3.6.1.4.1.5 - ISipTlsContextFactory::GetTlsClientContextS Method

Gets the TLS client context corresponding to the remote peer host name.

C++

```
virtual mxt_result GetTlsClientContextS(IN const CString& rstrHostname, OUT CTlsContext& rTlsContext) const = 0;
```

Parameters

Parameters	Description
IN const CString& rstrHostname	The remote peer host name.
OUT CTlsContext& rTlsContext	The TLS client context.

Returns

- resS_OK: The TLS client context has been found and returned in rTlsContext.
- resFE_INVALID_STATE: No TLS client context has been found for the specified remote peer host name.

- resFE_FAIL: Unable to process the message.

Description

Gets the TLS client context corresponding to the remote peer host name.

See Also

AddTlsClientContextS (see page 462), UpdateTlsClientContextS (see page 466), RemoveTlsClientContextS (see page 464).

3.6.1.4.1.6 - ISipTlsContextFactory::GetTlsServerContextS Method

Gets the TLS server context corresponding to the local listening address.

C++

```
virtual mxt_result GetTlsServerContextS(IN const CSocketAddr& rAddr, OUT CTlsContext& rTlsContext) const = 0;
```

Parameters

Parameters	Description
OUT CTlsContext& rTlsContext	The TLS server context.
rLocalAddr	The local listening address.

Returns

- resS_OK: The TLS server context has been found and returned in rTlsContext.
- resFE_INVALID_STATE: No TLS server context has been found for the specified local listening address.
- resFE_FAIL: Unable to process the message.

Description

Gets the TLS server context corresponding to the local listening address.

See Also

AddTlsServerContextS (see page 462), UpdateTlsServerContextS (see page 466), RemoveTlsServerContextS (see page 465).

3.6.1.4.1.7 - ISipTlsContextFactory::RemoveTlsClientContextS Method

Removes the TLS client context identified by the remote peer host name.

C++

```
virtual mxt_result RemoveTlsClientContextS(IN const CString& rstrHostname) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrHostname	The remote peer host name.

Returns

- resS_OK: The TLS client context is properly removed.
- resFE_INVALID_STATE: No TLS client context exists for the specified host name.
- resFE_FAIL: Unable to process message.

Description

Removes the TLS client context corresponding to the remote peer host name. If there are existing client TLS connections to the specified host name, these connections will be modified to use the default client context. It is then up to the application to initiate a renegotiation if desired.

See Also

AddTlsClientContextS (see page 462), UpdateTlsClientContextS (see page 466), GetTlsClientContextS (see page 463)

3.6.1.4.1.8 - ISipTlsContextFactory::RemoveTlsServerContextS Method

Removes the TLS server context identified by the remote peer socket address.

C++

```
virtual mxt_result RemoveTlsServerContextS(IN const CSocketAddr& rLocalAddr) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rLocalAddr	The local listening address.

Returns

- resS_OK: The TLS server context is properly removed.
- resFE_INVALID_STATE: No TLS server context exists for the specified local listening address.
- resFE_FAIL: Unable to process message.

Description

Removes the TLS server context corresponding to the remote peer socket address. If there are existing accepted TLS connections using the specified local listening address, these connections will be modified to use the default server context. It is then up to the application to initiate a renegotiation if desired.

See Also

AddTlsServerContextS (see page 462), UpdateTlsServerContextS (see page 466), GetTlsServerContextS (see page 464)

3.6.1.4.1.9 - ISipTlsContextFactory::SetDefaultTlsClientContextS Method

Sets the default TLS client context.

C++

```
virtual mxt_result SetDefaultTlsClientContextS(IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CTlsContext& rTlsContext	The default TLS client context.

Returns

- resS_OK: The default TLS server context is returned in the rTlsContext parameter.
- resFE_FAIL: Unable to process the message.

Description

Sets the default TLS client context. This is the generic context that applies to an outgoing TLS connection request when there is no specific context for a remote peer host name. If there are existing TLS connections that are using the default TLS client context, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

GetDefaultTlsClientContextS (see page 462)

3.6.1.4.1.10 - ISipTlsContextFactory::SetDefaultTlsServerContextS Method

Sets the default TLS server context.

C++

```
virtual mxt_result SetDefaultTlsServerContextS(IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CTlsContext& rTlsContext	The default TLS server context.

Returns

- resS_OK: The default TLS server context is returned in the rTlsContext parameter.
- resFE_FAIL: Unable to process the message.

Description

Sets the default TLS server context. This is the generic context that applies to an incoming TLS connection request when there is no specific context for a local listening address. If there are existing TLS connections that are using the default TLS server context, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

[GetDefaultTlsServerContextS](#) (see page 463)

3.6.1.4.1.11 - ISipTlsContextFactory::UpdateTlsClientContextS Method

Updates the TLS client context identified by the remote peer host name.

C++

```
virtual mxt_result UpdateTlsClientContextS(IN const CString& rstrHostname, IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrHostname	The remote peer host name.
IN const CTlsContext& rTlsContext	The TLS client context.

Returns

- resS_OK: The TLS client context is properly updated.
- resFE_INVALID_STATE: No TLS client context exists for the specified host name. Use [AddTlsClientContextS](#) (see page 462) to add a TLS client context for a new host name.
- resFE_FAIL: Unable to process message.

Description

Updates the TLS client context corresponding to the remote peer host name. If there are existing client TLS connections to the specified host name, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

[AddTlsClientContextS](#) (see page 462), [RemoveTlsClientContextS](#) (see page 464), [GetTlsClientContextS](#) (see page 463)

3.6.1.4.1.12 - ISipTlsContextFactory::UpdateTlsServerContextS Method

Updates the TLS server context identified by the remote peer socket address.

C++

```
virtual mxt_result UpdateTlsServerContextS(IN const CSocketAddr& rLocalAddr, IN const CTlsContext& rTlsContext) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rLocalAddr	The local listening address.
IN const CTlsContext& rTlsContext	The TLS server context.

Returns

- resS_OK: The TLS server context is properly updated.
- resFE_INVALID_STATE: No TLS server context exists for the specified socket address. Use AddTlsServerContextS (see page 462) to add a TLS server context for a new socket address.
- resFE_FAIL: unable to process message.

Description

Updates the TLS server context corresponding to the local listening address. If there are existing accepted TLS connections using the specified local listening address, these connections will be modified with the new context. It is then up to the application to initiate a renegotiation if desired.

See Also

AddTlsServerContextS (see page 462), RemoveTlsServerContextS (see page 465), GetTlsServerContextS (see page 464)

3.6.1.5 - ISipTransportObserver Class**Class Hierarchy**

C++

```
class ISipTransportObserver;
```

Description

This interface defines the necessary APIs used by the transport manager to notify interested modules of various events detected on the SIP transport module.

Location

SipTransport/ISipTransportObserver.h

See Also

ISipTransportMgr

Methods

Method	Description
•   EvConnectionClosed (see page 468)	Callback used by the transport manager to report the graceful termination of a connection.
•   EvConnectionEstablished (see page 468)	Callback used by the transport manager to report the successful creation of a persistent connection.
•   EvObserverRemoved (see page 468)	Callback used by the transport manager to report that the transport observer has been removed from its internal list and will no longer be notified of any event as a result.
•   EvPacketReceived (see page 468)	Callback used by the transport manager when a SIP packet is received.
•   EvTransportError (see page 469)	Callback used by the transport manager when a network level error is detected.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
EClosureType (see page 469)	
EInsertObserverPriority (see page 469)	

3.6.1.5.1 - Methods

3.6.1.5.1.1 - ISipTransportObserver::EvConnectionClosed Method

Callback used by the transport manager to report the graceful termination of a connection.

C++

```
virtual void EvConnectionClosed(IN const CSocketAddr& rSource, IN const CSocketAddr& rDestination, IN
ESipTransport eTransport, IN EClosureType eClosureType) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rSource	The source address of the connection.
IN const CSocketAddr& rDestination	The peer address of the connection.
IN ESipTransport eTransport	The associated transport.
IN EClosureType eClosureType	eLOCAL if locally initiated eREMOTE if initiated by the peer

Description

This event is reported by the transport manager after a socket is closed either locally or by the peer.

3.6.1.5.1.2 - ISipTransportObserver::EvConnectionEstablished Method

Callback used by the transport manager to report the successful creation of a persistent connection.

C++

```
virtual void EvConnectionEstablished(IN const CSocketAddr& rSource, IN const CSocketAddr& rDestination, IN
ESipTransport eTransport) = 0;
```

Parameters

Parameters	Description
IN const CSocketAddr& rSource	The source address of the connection.
IN const CSocketAddr& rDestination	The peer address of the connection.
IN ESipTransport eTransport	The associated transport.

Description

This event is reported by the transport manager after a socket is successfully created for a persistent connection.

3.6.1.5.1.3 - ISipTransportObserver::EvObserverRemoved Method

Callback used by the transport manager to report that the transport observer has been removed from its internal list and will no longer be notified of any event as a result.

C++

```
virtual void EvObserverRemoved() = 0;
```

Description

This event is reported by the transport manager when the transport manager shuts down or is asked to remove a transport manager from its internal list. Therefore, it indicates that the transport observer has been removed from its list and will no longer be notified of any upcoming event as a result.

3.6.1.5.1.4 - ISipTransportObserver::EvPacketReceived Method

Callback used by the transport manager when a SIP packet is received.

C++

```
virtual mxt_result EvPacketReceived(IN const CSipPacket& rPacket, IN bool bHandled) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The received SIP packet.
IN bool bHandled	True when the packet is handled by a previously called observer, false otherwise.

Returns

Must return resSW_SIPTRANSPORT_PACKET_UNHANDLED if the implementation did not handle the packet.

Must return resS_OK if the implementation has accepted to handle the supplied packet. The implementation of this event must never return resS_OK when bHandled is true.

Description

Event reporting the reception of a SIP packet to all observers. bHandled is used to let the observers after the one that actually manages the incoming packet know that a packet has been received, but they do not need to handle it.

3.6.1.5.1.5 - ISipTransportObserver::EvTransportError Method

Callback used by the transport manager when a network level error is detected.

C++

```
virtual void EvTransportError(IN mxt_result res, IN const CSocketAddr& rSource, IN const CSocketAddr& rDestination, IN ESipTransport eTransport) = 0;
```

Parameters

Parameters	Description
IN mxt_result res	The result corresponding to the actual network error.
IN const CSocketAddr& rSource	Source address of the socket in error.
IN const CSocketAddr& rDestination	Remote host where the socket was connected or trying to connect to.
IN ESipTransport eTransport	Associated transport.

Description

This event is generated when a network error is detected. The connection upon which the error occurred is automatically closed when the error is detected.

3.6.1.5.2 - Enumerations**3.6.1.5.2.1 - ISipTransportObserver::EClosureType Enumeration**

```
enum EClosureType {
    eLOCAL,
    eREMOTE
};
```

Description

Indicates the closure type.

Members

Members	Description
eLOCAL	Closed locally.
eREMOTE	Closed remotely.

3.6.1.5.2.2 - ISipTransportObserver::EInsertObserverPriority Enumeration

```
enum EInsertObserverPriority {
    eLOW_PRIORITY_OBSERVER,
    eHIGH_PRIORITY_OBSERVER
};
```

Description

Indicates the priority of the observer.

Members

Members	Description
eLOW_PRIORITY_OBSERVER	Observer will be added in low priority.
eHIGH_PRIORITY_OBSERVER	Observer will be added in high priority.

3.6.1.6 - ISipTransportUser Class

Class Hierarchy

ISipTransportUser

C++

```
class ISipTransportUser;
```

Description

The SIP transport user interface defines the event that can be reported back to modules using the transport manager.

Location

SipTransport/ISipTransportUser.h

See Also

ISipTransportMgr

Methods

Method	Description
EvCommandResult (see page 470)	Notifies the transport user of the result of an operation.

Legend

	Method
	abstract

3.6.1.6.1 - Methods

3.6.1.6.1.1 - ISipTransportUser::EvCommandResult Method

Notifies the transport user of the result of an operation.

C++

```
virtual void EvCommandResult(IN mxt_result res, IN mxt_opaque opq) = 0;
```

Parameters

Parameters	Description
IN mxt_result res	The result code. For further details, see the async method reporting this event upon completion or error.
IN mxt_opaque opq	Opaque parameter given to the transport manager.

Description

This event is reported by the transport manager to a specific user when the transport has completed an operation, successfully or not.

See Also

ISipTransportMgr async methods

3.7 - SipUserAgent

This section documents the Sources/SipUserAgent folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes ([see page 471](#))
- Variables ([see page 658](#))

3.7.1 - Classes

This section documents the classes of the Sources/SipUserAgent folder.

Classes

Class	Description
ISipDigestClientAuthMgr (see page 472)	
ISipDigestClientAuthSvc (see page 474)	
ISipGenericMgr (see page 479)	
ISipGenericSvc (see page 481)	
ISipGlareMgr (see page 483)	
ISipGlareSvc (see page 484)	
ISipMwiMgr (see page 485)	
ISipMwiSvc (see page 489)	
ISipNotifierMgr (see page 492)	
ISipNotifierSvc (see page 497)	
ISipOptionTagsMgr (see page 504)	
ISipPrivacyMgr (see page 505)	
ISipPrivacySvc (see page 507)	
ISipPublishMgr (see page 513)	
ISipPublishSvc (see page 516)	
ISipRedirectionMgr (see page 522)	
ISipRedirectionSvc (see page 523)	
ISipRefereeMgr (see page 527)	
ISipRefereeSvc (see page 531)	
ISipReferrerMgr (see page 535)	
ISipReferrerSvc (see page 542)	
ISipRegistrationMgr (see page 546)	
ISipRegistrationSvc (see page 548)	
ISipReliableProvisionalResponseMgr (see page 559)	
ISipReliableProvisionalResponseSvc (see page 562)	
ISipReplacesMgr (see page 566)	
ISipReplacesSvc (see page 568)	
ISipSessionMgr (see page 569)	
ISipSessionSvc (see page 578)	
ISipSessionTimerMgr (see page 582)	
ISipSessionTimerSvc (see page 584)	
ISipSubscriberMgr (see page 590)	
ISipSubscriberSvc (see page 595)	
ISipTransactionCompletionMgr (see page 604)	
ISipTransactionCompletionSvc (see page 605)	
ISipTransferMgr07 (see page 607)	
ISipTransferSvc07 (see page 614)	
ISipUaAssertedIdentityMgr (see page 623)	
ISipUaAssertedIdentitySvc (see page 628)	
ISipUpdateMgr (see page 633)	
ISipUpdateSvc (see page 637)	
ISipUserAgentSvc (see page 639)	

3.7.1.1 - ISipDigestClientAuthMgr Class

Class Hierarchy

ISipDigestClientAuthMgr

C++

```
class ISipDigestClientAuthMgr;
```

Description

The ISipDigestClientAuthMgr interface is the interface through which the ISipDigestClientAuthSvc (see page 474) reports events to the application. The SIP stack uses this interface to inform the application that the user is being challenged by a remote party to authenticate itself.

Location

SipUserAgent/ISipDigestClientAuthMgr.h

See Also

ISipDigestClientAuthSvc (see page 474)

Methods

Method	Description
◆ A EvAuthLoop (see page 472)	The service has detected a possible authentication loop.
◆ A EvCredentialsExist (see page 473)	One or more servers requested authentication and the service already has corresponding credentials.
◆ A EvCredentialsRequired (see page 473)	One or more remote servers requested authentication and the service does not have the corresponding credentials.
◆ A EvInvalidCredentials (see page 474)	The service has detected credentials that are always rejected.

Legend

◆	Method
A	abstract

3.7.1.1.1 - Methods

3.7.1.1.1.1 - ISipDigestClientAuthMgr::EvAuthLoop Method

The service has detected a possible authentication loop.

C++

```
virtual void EvAuthLoop(IN ISipDigestClientAuthSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthSvc* pSvc	The service that has detected the authentication loop.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The 401 or 407 response that contains the challenge(s).

Description

The service has detected too many consecutive challenges that may lead to a possible authentication loop. This situation can occur with multiple proxies that have short nonce lifetime.

When this situation occurs, the manager stops trying to send this request by calling ISipClientEventControl::CallNextClientEvent (see page 19).

If the manager was caching the user credentials and using them throughout more than one context, it should clear its cache for all the realms that were challenging the current request. Use ISipDigestClientAuthSvc::GetChallenges (see page 477) to access this list.

See Also

ISipDigestClientAuthSvc::GetChallenges (see page 477), ISipDigestClientAuthSvc::SetLoopThreshold (see page 477).

3.7.1.1.1.2 - ISipDigestClientAuthMgr::EvCredentialsExist Method

One ore more servers requested authentication and the service already has corresponding credentials.

C++

```
virtual void EvCredentialsExist(IN ISipDigestClientAuthSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthSvc* pSvc	The service that has detected the challenge and will be handling it.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The 401 or 407 response that contains the challenge(s).

Description

Informs the application that one or more remote servers have requested authentication from the local user agent.

This event is reported when the service has the corresponding credentials for all the challenges received. It is possible that the credentials currently used by the service are invalid, however depending on the network setup, it may be impossible to detect invalid credentials. Thus, the manager should call ISipClientEventControl::RelssueRequest (see page 20)() and rely on EvInvalidCredentials (see page 474) and EvAuthLoop (see page 472) for detecting authentication problems.

See Also

ISipClientEventControl::RelssueRequest (see page 20)

3.7.1.1.1.3 - ISipDigestClientAuthMgr::EvCredentialsRequired Method

One or more remote servers requested authentication and the service does not have the corresponding credentials.

C++

```
virtual void EvCredentialsRequired(IN ISipDigestClientAuthSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthSvc* pSvc	The service that has detected the challenge and will be handling it.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The 401 or 407 response that contains the challenge(s).

Description

Informs the application that one or more remote servers have requested authentication from the local user agent.

This event is reported when the service has no credentials for one or more challenges received. The manager must therefore provide authentication information for the request to progress further.

The manager can access the list of challenges by using the ISipDigestClientAuthSvc::GetChallenges (see page 477) method. This list contains ALL challenges received, including the ones for which the service may have valid credentials. The m_strHashA1 member of SChallengeData may be used to find the challenges for which there are no available credentials yet. Calling m_strHashA1.IsEmpty() returns true for challenges for which the service has no credentials.

Once the manager has obtained the credentials for a challenge, it must use the ISipDigestClientAuthSvc::Authenticate (see page 476) method to configure the credentials. This should be called once for each challenges for which the service does not have corresponding credentials.

After all credentials are provided, the manager should use ISipClientEventControl::RelssueRequest (see page 20) to retry the request with the new credentials.

See Also

[ISipDigestClientAuthSvc::GetChallenges](#) (see page 477), [ISipDigestClientAuthSvc::Authenticate](#) (see page 476), [ISipClientEventControl::ReissueRequest](#) (see page 20)

3.7.1.1.1.4 - ISipDigestClientAuthMgr::EvInvalidCredentials Method

The service has detected credentials that are always rejected.

C++

```
virtual void EvInvalidCredentials(IN ISipDigestClientAuthSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CString& rstrRealm, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthSvc* pSvc	The service that has detected the challenge and will be handling it.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CString& rstrRealm	The realm for which the credentials are invalid.
IN const CSipPacket& rResponse	The 401 or 407 response that contains the challenge(s).

Description

The authentication service has detected that the currently configured credentials for the realm rstrRealm are invalid. This situation is detected when the service receives a challenge twice in a row from the same realm.

The manager should either provide new credentials for the specified realm or call [ISipClientEventControl::CallNextClientEvent](#) (see page 19)() to abandon trying to send the request.

See Also

[ISipClientEventControl::CallNextClientEvent](#) (see page 19), [ISipDigestClientAuthSvc::Authenticate](#) (see page 476)

3.7.1.2 - ISipDigestClientAuthSvc Class**Class Hierarchy****C++**

```
class ISipDigestClientAuthSvc : public IEComUnknown;
```

Description

This service helps the application authenticate itself to a remote party that challenges it. It monitors transactions for 401 (Unauthorized) and 407 (Proxy Authentication Required) responses and then generates events to the application when required. It supports Digest authentication.

Adding an ISipDigestClientAuthSvc to a SIP context makes it possible to easily respond to challenges for any SIP request, even one that does not exist yet!

If an application chooses not to add this service to a context, then it is up to the application to detect challenges and to properly build the credentials for the next packet to be sent. This interface simply makes this process much easier.

This service generates events to the application through the [ISipDigestClientAuthMgr](#) (see page 472) interface.

The ISipDigestClientAuthMgr is an ECOM object

The ISipDigestClientAuthMgr (see page 472) is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipDigestClientAuthMgr

Interface Id: IID_ISipDigestClientAuthMgr

A user can query the ISipContext (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all

other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipDigestClientAuthSvc.h

See Also

ISipDigestClientAuthMgr (see page 472)

Methods

Method	Description
◆ A Authenticate (see page 476)	Sets the user name and hash of A1 for a specific realm.
◆ A EnableQualityOfProtectionAuthInt (see page 476)	Configures the quality of protection sent in challenges.
◆ A GetChallenges (see page 477)	Gets the challenges that currently require authentication, if any.
◆ A Reset (see page 477)	Clears the authentication state.
◆ A SetLoopThreshold (see page 477)	Configures the maximum number of consecutive challenges the service may handle before reporting EvAuthLoop.
◆ A SetManager (see page 478)	Configures the event manager associated with this service manager.
◆ A SetRouteUriAuthentication (see page 478)	Sets the service's behaviour for using the route URI instead of the request URI.

Legend

◆	Method
◆ A	abstract

Enumerations

Enumeration	Description
EUnknownQopBehavior (see page 478)	

Structs

Struct	Description
SChallengeData (see page 475)	

3.7.1.2.1 - Structs

3.7.1.2.1.1 - ISipDigestClientAuthSvc::SChallengeData Struct

```
struct SChallengeData {
    CString m_strRealm;
    CString m_strOpaque;
    CString m_strNonce;
    CString m_strAlgorithm;
    CString m_strQop;
    CString m_strUsername;
    CString m_strHashA1;
    CString m_strCNonce;
    CString m_strAuthHeader;
    unsigned int m_uNonceCount;
    unsigned int m_uChallengeType;
    bool m_bQopAuthSupported;
    bool m_bQopAuthIntSupported;
    bool m_bSuccessfullyAuthenticated;
};
```

Description

This structure contains parameters used in Proxy-Authenticate (see page 476), WWW-Authenticate (see page 476), Proxy-Authorization and Authorization headers.

Members

Members	Description
CString m_strRealm;	realm" parameter.
CString m_strOpaque;	opaque" parameter.

CString m_strNonce;	nonce" parameter.
CString m_strAlgorithm;	algorithm" parameter.
CString m_strQop;	qop" parameter.
CString m_strUsername;	username" parameter.
CString m_strHashA1;	Hashing of username, realm and password used in "response" parameter.
CString m_strCNonce;	cnonce" parameter.
CString m_strAuthHeader;	For a Proxy-Authorization or Authorization header, all the digest-response parameters of the header.
unsigned int m_uNonceCount;	nc" parameter.
unsigned int m_uChallengeType;	Challenge type (user agent, proxy).
bool m_bQopAuthSupported;	Whether "auth" is included in qop.
bool m_bQopAuthIntSupported;	Whether "auth-int" is included in qop.
bool m_bSuccessfullyAuthenticated;	Whether authentication succeeded.

3.7.1.2.2 - Methods

3.7.1.2.2.1 - ISipDigestClientAuthSvc::Authenticate Method

Sets the user name and hash of A1 for a specific realm.

C++

```
virtual mxt_result Authenticate(IN const CString& rstrRealm, IN const CString& rstrUsername, IN const CString& rstrHashA1) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrRealm	The realm for which the user name and hash of A1 are set.
IN const CString& rstrHashA1	The hash of A1 containing the password corresponding to that user name.
rstrUserName	The user name that should be used to authenticate the user.

Returns

resFE_FAIL: No data was present for this realm.

resFE_DUPLICATE: Credentials were already present for this realm, which means they had been set previously and possibly a request was sent containing them.

resS_OK: Credentials were added successfully for this realm.

Description

This must be called by the application to set authentication information after the service manager has generated the EvAuthenticationRequired event.

It configures the client authentication service with the user name and hash of A1 it should use to authenticate the user at the remote party.

It returns an error if there was no challenge received in the parent context or a warning if the user name and hash of A1 had previously been tried unsuccessfully.

Every future requests in the parent context will include credentials based on these user name and hash of A1 until a new user name and a new hash of A1 are set through this method.

After configuring the service manager with a user name and a hash of A1, the application must re-send the request that was challenged, which will then contain the proper credentials.

3.7.1.2.2.2 - ISipDigestClientAuthSvc::EnableQualityOfProtectionAuthInt Method

Configures the quality of protection sent in challenges.

C++

```
virtual void EnableQualityOfProtectionAuthInt(IN bool bEnable) = 0;
```

Parameters

Parameters	Description
IN bool bEnable	A boolean to enable or disable the Auth-Int authentication method.

Description

Configures the usage of the Auth-Int authentication method.

As explained in RFC 2617, disabling the Auth-Int will change the way that A2 is computed.

Also, if a challenge has qop=auth,auth-int and auth-int is disabled, the client will only set a qop=auth in the Authorization header.

3.7.1.2.2.3 - ISipDigestClientAuthSvc::GetChallenges Method

Gets the challenges that currently require authentication, if any.

C++

```
virtual const CVector<SChallengeData*>& GetChallenges() const = 0;
```

Returns

Challenges that currently require authentication.

Description

Gets the challenges that currently require authentication, if any. Must not be called if no challenge was received. The application is informed via an event EvAuthRequired or EvChallengedAfterSuccess on interface ISipDigestClientAuthMgr (see page 472) that a challenge was received.

3.7.1.2.2.4 - ISipDigestClientAuthSvc::Reset Method

Clears the authentication state.

C++

```
virtual void Reset() = 0;
```

Description

The client authentication service automatically reuses the last successful credentials for any new request that is sent. In some circumstances, this behaviour is not desirable.

The Reset() method clears any credential, hash of A1 and authentication state that the service keeps. This has the effect that new requests sent on an already authenticated dialog will not contain any credentials.

The best moment to Reset() the credentials is before sending a new request.

3.7.1.2.2.5 - ISipDigestClientAuthSvc::SetLoopThreshold Method

Configures the maximum number of consecutive challenges the service may handle before reporting EvAuthLoop.

C++

```
virtual mxt_result SetLoopThreshold(unsigned int uMax) = 0;
```

Parameters

Parameters	Description
unsigned int uMax	The maximum number of consecutive challenges the service may handle.

Returns

- resFE_INVALID_ARGUMENT: uMax is too low. Must be higher than 2.
- resS_OK: Maximum correctly configured.

Description

This configures the maximum number of consecutive 401/407 challenges the service may handle before reporting the EvAuthLoop event to the manager. 401/407 challenges are considered consecutive when they are not separated by a success response (2xx).

A loop is detected when the uMax consecutive 401/407 challenges are received. For instance, if uMax is 10, when the 10th 401/407 response is received since either the first or last 2xx response, the EvAuthLoop event is issued.

The default value is 10 when this configuration method is not called.

3.7.1.2.2.6 - ISipDigestClientAuthSvc::SetManager Method

Configures the event manager associated with this service manager.

C++

```
virtual mxt_result SetManager(IN ISipDigestClientAuthMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipDigestClientAuthMgr* pMgr	The event manager. Must not be NULL.

Description

Configures the event manager that receives the events generated by this service manager.

Note that an event manager MUST be associated with this service manager before it is used.

3.7.1.2.2.7 - ISipDigestClientAuthSvc::SetRouteUriAuthentication Method new in 4.1.4

Sets the service's behaviour for using the route URI instead of the request URI.

C++

```
virtual void SetRouteUriAuthentication(IN bool bUseRouteUri) = 0;
```

Parameters

Parameters	Description
IN bool bUseRouteUri	True to have the authentication service use the route.

Description

Sets the service's behaviour for choosing which URI it will use in the Proxy-Authorization header. If false, the service will use the default request URI of the SIP packet before any route headers are applied to it. This is the default behaviour. If set to true the URI in the Proxy-Authorization header will contain the URI set in the first Route header. When no route header can be found and bUseRouteUri is true, the default behaviour will be used. That is the packet's REQUEST-URI will be used to perform the authentication.

3.7.1.2.3 - Enumerations

3.7.1.2.3.1 - ISipDigestClientAuthSvc::EUnknownQopBehavior Enumeration

```
enum EUnknownQopBehavior {
    eFAIL_ON_UNKNOWN_QOP,
    eIGNORE_UNKNOWN_QOP
};
```

Description

The different possible behaviors of the ISipDigestClientAuthSvc (see page 474) upon reception of an unknown qop parameter value in the WWW-Authenticate (see page 476) or Proxy-Authenticate (see page 476) header.

Members

Members	Description
eFAIL_ON_UNKNOWN_QOP	Upon reception of an unknown qop value the request will fail and a re-issued request will not contain any response for the received challenge. This is the default behavior.
eIGNORE_UNKNOWN_QOP	Upon reception of an unknown qop value the ISipDigestClientAuthSvc (see page 474) will act exactly as if it did not receive any qop parameter value in the WWW-Authenticate (see page 476) or Proxy-Authenticate (see page 476) header. This means that when the request is reissued the challenge response will be generated as if no qop parameter was present in the challenge. Please note the following RFC quotations before the behavior is chosen: RFC 2617 section 3.2.1 states that: "Unrecognized options MUST be ignored." While RFC3261 section 22.4 bullet 8 states that: "If a client receives a "qop" parameter in a challenge header field, it MUST send the "qop" parameter in any resulting authorization header field."

3.7.1.3 - ISipGenericMgr Class

Class Hierarchy

ISipGenericMgr

C++

```
class ISipGenericMgr;
```

Description

The generic manager is the interface through which the generic service reports events to the application. The generic service informs the application through this interface of incoming generic requests and incoming responses to generic requests sent to a remote party.

Location

SipUserAgent/ISipGenericMgr.h

See Also

ISipGenericSvc (see page 481)

Methods

Method	Description
◆ A EvFailure (see page 479)	Reports that a final response (>= 300) to a previously sent request has been received.
◆ A EvProgress (see page 480)	Reports that a provisional response to a previously sent request has been received.
◆ A EvRequest (see page 480)	Reports that a SIP request has been received.
◆ A EvSuccess (see page 480)	Reports that a final response (2xx) to a previously sent request has been received.

Legend

◆	Method
◆ A	abstract

3.7.1.3.1 - Methods

3.7.1.3.1.1 - ISipGenericMgr::EvFailure Method

Reports that a final response (>= 300) to a previously sent request has been received.

C++

```
virtual void EvFailure(IN ISipGenericSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipGenericSvc* pSvc	The ISipGenericSvc (see page 481) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	The interface to the client event control for this transaction.

IN const CSipPacket& rResponse

The SIP response.

Description

Informs the manager that a response to one of the generic request it has sent has been received. This event can be called for one final response (>= 300).

See Also

[EvProgress](#) (see page 480), [EvSuccess](#) (see page 480), [ISipGenericSvc::SendRequest](#) (see page 482)

3.7.1.3.1.2 - ISipGenericMgr::EvProgress Method

Reports that a provisional response to a previously sent request has been received.

C++

```
virtual void EvProgress(IN ISipGenericSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipGenericSvc* pSvc	The ISipGenericSvc (see page 481) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	The interface to the client event control for this transaction.
IN const CSipPacket& rResponse	The SIP response.

Description

Informs the manager that a response to one of the generic request it has sent has been received. This event can be called for zero to many provisional responses (1xx).

See Also

[EvSuccess](#) (see page 480), [EvFailure](#) (see page 479), [ISipGenericSvc::SendRequest](#) (see page 482)

3.7.1.3.1.3 - ISipGenericMgr::EvRequest Method

Reports that a SIP request has been received.

C++

```
virtual void EvRequest(IN ISipGenericSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipGenericSvc* pSvc	The ISipGenericSvc (see page 481) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The SIP request.

Description

Informs the manager that a generic request has been received. The application MUST use the [SendResponse](#) method of the [ISipServerEventControl](#) (see page 75) in parameter to send a response to that request.

Managers should not send 1xx responses to non-INVITE requests, but should instead answer as soon as possible with a final response.

See Also

[ISipServerEventControl::SendResponse](#) (see page 76)

3.7.1.3.1.4 - ISipGenericMgr::EvSuccess Method

Reports that a final response (2xx) to a previously sent request has been received.

C++

```
virtual void EvSuccess(IN ISipGenericSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipGenericSvc* pSvc	The ISipGenericSvc (see page 481) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	The interface to the client event control for this transaction.
IN const CSipPacket& rResponse	The SIP response.

Description

Informs the manager that a response to one of the generic request it has sent has been received. This event can be called for one final response (2xx).

See Also

EvProgress (see page 480), EvFailure (see page 479), ISipGenericSvc::SendRequest (see page 482)

3.7.1.4 - ISipGenericSvc Class**Class Hierarchy****C++**

```
class ISipGenericSvc : public IEComUnknown;
```

Description

The generic service offers the possibility to send and receive any type of SIP request and their responses. It can be used to send multiple requests and receive their responses simultaneously. Generic requests are requests that follow the simple request/response pattern, as opposed to request/response/ACK. This means it can be used to manage non-INVITE and non-ACK requests.

It SHOULD NOT be used to manage INVITEs on the UAS side, the session service SHOULD be preferred for such a task. The generic service is not tailored to manage all the complexities of received INVITEs.

This service is used for sending requests that are not directly supported by other services such as OPTIONS, INFO, MESSAGE, etc. The generic service can also be used for managing other types of requests that are already managed by an existing service, such as UPDATE, REFER, or REGISTER. However this is not usually recommended, unless the user has an in-depth knowledge of the requirements for the other types of requests.

Even though the BYE request fits the "non-INVITE" request definition, it should only be sent through the ISipSessionSvc (see page 578) in order for the Session Service to maintain proper dialog state.

The Generic Service SendRequest (see page 482) method must **NOT** be used to send CANCEL requests. Use the ISipClientTransaction::CancelRequest (see page 22) method from the transaction to cancel instead.

This service generates events to the application through the ISipGenericMgr (see page 479) interface.

The ISipGenericSvc is an ECOM object

The ISipGenericSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipGenericSvc

Interface Id: IID_ISipGenericSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipGenericSvc.h

See Also

ISipGenericMgr (see page 479), ISipContext (see page 23)

Methods

Method	Description
• A SendRequest (see page 482)	Sends the specified request.
• A SetManager (see page 482)	Configures the manager that will receive events from this service.

Legend

•	Method
A	abstract

3.7.1.4.1 - Methods**3.7.1.4.1.1 - ISipGenericSvc::SendRequest Method**

Sends the specified request.

C++

```
virtual mxt_result SendRequest(IN const char* szMethod, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const char* szMethod	The SIP method to send.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	The SIP headers to add in the packet. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	The packet's body, aka payload. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Reference to the created client transaction.

Returns

resS_OK: Request is being sent. Feedback is provided as a response to the request through the service manager.

resFE_FAIL: Unable to send the request. No other feedback is provided for this request.

Description

Sends the request over the network with the specified SIP headers and payload when present.

The reference to ISipClientTransaction (see page 21) allows users to further control the transaction. It is possible to cancel a generic request by using ISipClientTransaction::CancelRequest (see page 22); however, this is not recommended. RFCs 4320 and 4321 highlight the problems related to cancelling a non-INVITE request.

See Also

ISipGenericSvc::EvResponse, ISipClientTransaction (see page 21),

3.7.1.4.1.2 - ISipGenericSvc::SetManager Method

Configures the manager that will receive events from this service.

C++

```
virtual mxt_result SetManager(IN ISipGenericMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipGenericMgr* pMgr	The manager. Must not be NULL.

Returns

resFE_FAIL: pMgr is NULL.

resS_OK: Otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that an event manager MUST be associated with this service before it is used.

3.7.1.5 - ISipGlareMgr Class**Class Hierarchy**

ISipGlareMgr

C++

```
class ISipGlareMgr;
```

Description

The Glare manager is the interface through which the Glare core service talks to the application. The application is informed through this interface that it is time to reissue a request that was in glare if it is still needed.

Location

SipUserAgent/ISipGlareMgr.h

See Also

ISipGlareSvc (see page 484)

Methods

Method	Description
● A EvReadyToRetry (see page 483)	Informs the application that it is time to retry a request that received a 491 Request Pending response.

Legend

●	Method
A	abstract

3.7.1.5.1 - Methods**3.7.1.5.1.1 - ISipGlareMgr::EvReadyToRetry Method**

Informs the application that it is time to retry a request that received a 491 Request Pending response.

C++

```
virtual void EvReadyToRetry(IN ISipGlareSvc* pGlareSvc, IN bool bOtherRequestSeenSinceGlare) = 0;
```

Parameters

Parameters	Description
IN ISipGlareSvc* pGlareSvc	Interface on which the glare occurred.
IN bool bOtherRequestSeenSinceGlare	True when new, non-ACK requests have been sent or received on the context since the 491 response was received, false otherwise.

Description

Informs the application that the last request that received a 491 response can be retried.

This event will be generated at a random time between 0 and 2 seconds after the reception of a 491 when the context is not the owner of its Call-ID. This is the case when the context was a UAS for the first request on it.

When the context is the owner of the Call-ID (it was a UAC on the first request in the context), the event will occur at a random time between 2.1 and 4 seconds.

On reception of a 491 response to a sent request:

- The service that issued the request generates a failure.

In the processing of the failure event, the application needs to identify that the response code is a 491 and take note of any information that can be relevant to the reissuing of the request at a later time. Examples of relevant information are the ISipClientControlEvent pointer and application state information.

- When EvReadyToRetry is generated, the application must decide if it still wants to send the request that glared. It can regenerate the request or optionally use the saved ISipClientEventControl::RelIssueRequest (see page 20) to do so. When bOtherRequestSeenSinceGlare is true, other requests have been exchanged on the context since the request in glare (the new requests may be related or not to the request in glare). In this case, it may be irrelevant to send the request again.

For instance, let's say that the application is sending a re-INVITE for a session timer refresh and that at the same time the remote is sending one to put the application on hold. The application generates a 491 for the received re-INVITE and the remote does the same. Assuming that the application is the owner of the Call-ID, the glare service starts a random timer between 2.1 to 4 seconds. While the local timer is elapsing, the remote retries its hold re-INVITE (its retry timer will be between 0 and 2 seconds). In this particular scenario, when the local timer fires and EvReadyToRetry is generated, there would be no point in sending session-timer refresh INVITE since the context would have been refreshed by the received hold re-INVITE.

3.7.1.6 - ISipGlareSvc Class

Class Hierarchy



C++

```
class ISipGlareSvc : public IEComUnknown;
```

Description

The Glare service handles the case where both parties send each other a request simultaneously. As per RFC 3261 section 21.4.27 and section 14.1, when the SIP stack receives a 491 (Request Pending), the Glare service starts a timer. When the timer fires, it is reported to the application through the generation of an event using the Glare manager. The application can then decide whether it wants to reissue the request or not.

In addition to the INVITE, this service acts on any non-ACK method.

Refer to the ISipGlareMgr (see page 483) documentation for more details about the events that this service can report.

Note that an event manager MUST be associated with this service before it is used. See the SetManager (see page 485) method for more information.

The ISipGlareSvc is an ECOM object

The ISipGlareSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipGlareSvc

Interface Id: IID_ISipGlareSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgentSvc/ISipGlareSvc.h

Methods

Method	Description
• A SetManager (see page 485)	Configures the event manager associated with this service.

Legend

	Method
	abstract

3.7.1.6.1 - Methods**3.7.1.6.1.1 - ISipGlareSvc::SetManager Method**

Configures the event manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipGlareMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipGlareMgr* pMgr	The event manager. Must not be NULL.

Description

Configures the event manager that will receive the events generated by this service.

Note that an event manager MUST be associated with this service before it is used.

3.7.1.7 - ISipMwiMgr Class**Class Hierarchy**

ISipMwiMgr

C++

```
class ISipMwiMgr;
```

Description

The Message Waiting Indication Event Manager is the interface through which the MWI service talks to the application. The SIP Stack informs the application through this interface of the progress for SUBSCRIBE requests and reception of NOTIFY requests.

Location

SipUserAgent/ISipMwiMgr.h

See Also

ISipMwiSvc (see page 489)

Methods

Method	Description
  EvExpired (see page 486)	The subscription has expired.
  EvExpiring (see page 486)	The subscription is about to expire.
  EvFailure (see page 486)	Notifies the application of the failure of an outgoing subscription.
  EvIntervalTooSmall (see page 486)	Notifies the application of the failure of an outgoing subscription because the interval is too small.
  EvInvalidNotify (see page 487)	Notifies the application that an invalid NOTIFY request has been received.
  EvNotified (see page 488)	Notifies the application of a change to the message-box's state.
  EvProgress (see page 488)	Notifies the application of the progress of an outgoing subscription.
  EvShutdown (see page 488)	Notifies the application that the message-box is about to be closed.
  EvSuccess (see page 489)	Notifies the application of the success of an outgoing subscription.

Legend

	Method
	abstract

3.7.1.7.1 - Methods

3.7.1.7.1.1 - ISipMwiMgr::EvExpired Method

The subscription has expired.

C++

```
virtual void EvExpired(IN ISipMwiSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.

Description

This event is reported when the subscription has expired. The application must no longer expect any notification about the state of the message-box.

3.7.1.7.1.2 - ISipMwiMgr::EvExpiring Method

The subscription is about to expire.

C++

```
virtual void EvExpiring(IN ISipMwiSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.

Description

This event is reported when the subscription is about to expire. This means that the application should refresh the subscription by re-subscribing.

3.7.1.7.1.3 - ISipMwiMgr::EvFailure Method

Notifies the application of the failure of an outgoing subscription.

C++

```
virtual void EvFailure(IN ISipMwiSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
IN const CSipPacket& rPacket	The failure final response received.

Description

Informs the application that a response was received for the SUBSCRIBE request that was just sent. When a response to a subscription refresh has a status code other than a 481, the subscription remains active and a refresh can still be retried until the subscription expires.

3.7.1.7.1.4 - ISipMwiMgr::EvIntervalTooSmall Method

Notifies the application of the failure of an outgoing subscription because the interval is too small.

C++

```
virtual void EvIntervalTooSmall(IN ISipMwiSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN unsigned int uMinExpirationSec, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
IN unsigned int uMinExpirationSec	The value of the Min-Expires header found in the 423 response.
IN const CSipPacket& rPacket	The response received.

Description

Notifies the application that a response indicating an interval too small has been received.

3.7.1.7.1.5 - ISipMwiMgr::EvInvalidNotify Method

Notifies the application that an invalid NOTIFY request has been received.

C++

```
virtual void EvInvalidNotify(IN ISipMwiSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rPacket, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rPacket	The NOTIFY request received.
IN mxt_result res	<ul style="list-style-type: none"> resFE_UNKNOWN_EVENT: The Event header was missing or for an event type other than "message-summary". A "489 Bad Event" response has already been sent. resFE_UNKNOWN_SUBSCRIPTION: The NOTIFY does not correspond to a REFER sent by this service. A "481 Subscription does not exist" response has already been sent. resFE_EXPIRED_SUBSCRIPTION: The NOTIFY has been received after another NOTIFY with "subscription-state" "terminated" or after the expiration time. A "481 Subscription does not exist" response has already been sent. resFE_MISSING_HEADER: The received NOTIFY did not contain the mandatory "Subscription-State" header. resFE_INVALID_CONTENT: The received NOTIFY did not contain a valid "application/simple-message-summary".

Description

Notifies the application that an invalid NOTIFY request has been received and what is the error code.

3.7.1.7.1.6 - ISipMwiMgr::EvNotified Method

Notifies the application of a change to the message-box's state.

C++

```
virtual void EvNotified(IN ISipMwiSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rPacket, IN TO CMesssageSummary* pMessageSummary) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rPacket	The NOTIFY request.
IN TO CMesssageSummary* pMessageSummary	The NOTIFY request's payload (message summary).

Description

Notifies the application that the state of the message-box has changed. This usually means that a message has been added or deleted from the message-box.

When receiving a NOTIFY with no payload and it is not the first NOTIFY, a 200 OK response is returned without this event being called.

3.7.1.7.1.7 - ISipMwiMgr::EvProgress Method

Notifies the application of the progress of an outgoing subscription.

C++

```
virtual void EvProgress(IN ISipMwiSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
IN const CSipPacket& rPacket	The provisional response received.

Description

Notifies the application that a response has been received for the SUBSCRIBE request that was just sent.

3.7.1.7.1.8 - ISipMwiMgr::EvShutdown Method

Notifies the application that the message-box is about to be closed.

C++

```
virtual void EvShutdown(IN ISipMwiSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rPacket, IN TO CMesssageSummary* pMessageSummary) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing MWI messages.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rPacket	The NOTIFY request.
IN TO CMesssageSummary* pMessageSummary	The NOTIFY request's payload (message summary).

Description

Notifies the application that the subscription has been terminated by the remote server because it went offline.

3.7.1.7.1.9 - ISipMwiMgr::EvSuccess Method

Notifies the application of the success of an outgoing subscription.

C++

```
virtual void EvSuccess(IN ISipMwiSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipMwiSvc* pSvc	The ISipMwiSvc (see page 489) managing Mwi messages.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
IN const CSipPacket& rPacket	The success final response received.

Description

Notifies the application that a response has been received for the SUBSCRIBE request that was just sent.

3.7.1.8 - ISipMwiSvc Class

Class Hierarchy



C++

```
class ISipMwiSvc : public IEComUnknown;
```

Description

The Message Waiting Indication service is used to subscribe to the state of a message-box account held by a server. When the state of the message-box account changes, whether because a message was added or removed, the subscriber is notified of such change. This allows the application to notify a user in some way that the state of his message-box has changed.

This service is the implementation of the "message-summary" event package for RFC 3265. As such, this service sends the SUBSCRIBE SIP request to subscribe to the state of a message-box and expects to receive NOTIFY requests when the message-box state has changed.

This service generates events to the application through the ISipMwiMgr (see page 485) interface.

The ISipMwiSvc is an ECOM object

The ISipMwiSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipMwiSvc

Interface Id: IID_ISipMwiSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipMwiSvc.h

See Also

CSipMwiSvc, ISipMwiMgr (see page 485)

Methods

Method	Description
• A SetExpiringThreshold (see page 490)	Sets the time to warn the application before a subscription expires.
• A SetManager (see page 490)	Configures the event manager associated with this service.
• A Subscribe (see page 490)	Subscribes to a message-box state.

✖ A Unsubscribe (see page 491)	Removes a subscription to a message-box state, or performs a Fetch if the subscription does not already exist.
--------------------------------	--

Legend

✖	Method
A	abstract

3.7.1.8.1 - Methods

3.7.1.8.1.1 - ISipMwiSvc::SetExpiringThreshold Method Updated behavior in 4.1.4

Sets the time to warn the application before a subscription expires.

C++

```
virtual void SetExpiringThreshold(IN unsigned int uThresholdSec) = 0;
```

Parameters

Parameters	Description
IN unsigned int uThresholdSec	The time, in seconds, before expiration time that the EvExpiring is fired.

Description

Sets the amount of time that should remain to a subscription before EvExpiring is called for that subscription. If not called, the default value is one minute (60 seconds).

This method affects only future subscriptions. This means that if there is an active subscription while this method is called, the EvExpiring event is called according to the previous threshold value.

If the expires value is smaller than or equal to the threshold, the EvExpiring event is never called, EvExpired is instead called directly.

See Also

Subscribe (see page 490), Unsubscribe (see page 491)

3.7.1.8.1.2 - ISipMwiSvc::SetManager Method

Configures the event manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipMwiMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipMwiMgr* pMgr	The event manager. Must not be NULL.

Description

Configures the event manager that will receive the events generated by this service.

Note that an event manager MUST be associated with this service before it is used.

3.7.1.8.1.3 - ISipMwiSvc::Subscribe Method

Subscribes to a message-box state.

C++

```
virtual mxt_result Subscribe(IN unsigned int uExpirationSec, IN mxt_opaque opqTransaction, OUT ISipClientTransaction*& rpTransaction, IN TO CHeaderList* pExtraHeaders = NULL) = 0;
```

Parameters

Parameters	Description
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When set to 0, the default expiration is used, which is 3600 seconds (one hour).
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
IN TO CHeaderList* pExtraHeaders = NULL	Extra SIP headers to send with the request. Can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.

Returns

resFE_INVALID_STATE: The event manager has not been set yet or Clear has been called on the context.

resFE_FAIL: The SUBSCRIBE request could not be sent.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK: The SUBSCRIBE request was successfully sent.

Description

This method is used to subscribe the state of a message-box. This has the effect of sending a SUBSCRIBE request to the SIP request-URI currently configured in the context to which this service is attached.

When an expiration of 0 is specified, an Unsubscribe (see page 491) or a Fetch is performed instead of a Subscribe or a Refresh.

Notification about the request's progress is received through the ISipMwiMgr (see page 485) interface.

Upon receiving a final negative response other than a 481 on a subscription refresh (i.e. Subscribe has been called while the subscription is still active and EvFailure has been called on the ISipMwiMgr (see page 485)), the subscription is still considered active and a refresh can therefore be retried until the subscription expires.

See Also

ISipMwiMgr (see page 485)

3.7.1.8.1.4 - ISipMwiSvc::Unsubscribe Method

Removes a subscription to a message-box state, or performs a Fetch if the subscription does not already exist.

C++

```
virtual mxt_result Unsubscribe(IN mxt_opaque opqTransaction, OUT ISipClientTransaction*& rpTransaction, IN TO CHeaderList* pExtraHeaders = NULL) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
IN TO CHeaderList* pExtraHeaders = NULL	Extra SIP headers to send with the request. Can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.

Returns

resFE_INVALID_STATE: The event manager has not been set yet or Clear has been called on the context.

resFE_FAIL: The SUBSCRIBE request could not be sent.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK: The SUBSCRIBE request is successfully sent.

Description

This method is used to remove a subscription to the state of a message-box. This has the effect of sending a SUBSCRIBE request to

the SIP request-URI currently configured in the context to which this service is attached, with an expiration value of zero.

Notification about the request's progress is received through the **ISipMwiMgr** (see page 485) interface.

See Also

ISipMwiMgr (see page 485)

3.7.1.9 - **ISipNotifierMgr** Class New in 4.1.4

Class Hierarchy

ISipNotifierMgr

C++

```
class ISipNotifierMgr;
```

Description

The interface through which the **ISipNotifierSvc** (see page 497) reports events to the application.

Warning

This class replaces the old class of the same name, which has been deprecated and renamed **ISipNotifierBaseMgr** for clarity.

Location

SipUserAgent/ISipNotifierMgr.h

See Also

ISipNotifierSvc (see page 497)

Methods

Method	Description
● A EvExpired (see page 492)	The subscription is expired.
● A EvFailure (see page 493)	A failure (<= 3xx) has been received for a NOTIFY request sent by ISipNotifierSvc (see page 497).
● A EvFetched (see page 493)	A SUBSCRIBE request to fetch the current state has been received.
● A EvInvalidSubscribe (see page 494)	A SUBSCRIBE request has been automatically rejected by this service.
● A EvProgress (see page 494)	A provisional response (1xx) has been received for a NOTIFY request sent by the ISipNotifierSvc (see page 497).
● A EvRefreshed (see page 495)	A SUBSCRIBE request to refresh an active subscription has been received.
● A EvSubscribed (see page 495)	A SUBSCRIBE request for a new subscription has been received.
● A EvSuccess (see page 496)	A success response (2xx) has been received for a NOTIFY request sent by the ISipNotifierSvc (see page 497).
● A EvTerminated (see page 496)	A SUBSCRIBE request to terminate an active/pending subscription has been received.

Legend

●	Method
■	abstract

3.7.1.9.1 - Methods

3.7.1.9.1.1 - **ISipNotifierMgr::EvExpired** Method

The subscription is expired.

C++

```
virtual void EvExpired(IN ISipNotifierSvc* pSvc, IN const CString& rstrEvent, IN const CString& rstrId) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface that has an expired subscription.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID of the expired subscription.

Description

Informs the application that a subscription is expired. This only happens for subscriptions created through CreateSubscription or by accepting a SUBSCRIBE request through EvSubscribed (see page 495) and not calling Terminate for that subscription.

When this event occurs, the application should call Terminate unless the subscription was implicit, in which case it may call Terminate.

3.7.1.9.1.2 - ISipNotifierMgr::EvFailure Method

A failure (<= 3xx) has been received for a NOTIFY request sent by ISipNotifierSvc (see page 497).

C++

```
virtual void EvFailure(IN ISipNotifierSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the NOTIFY request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must either call CallNextEvent, ClearClientEvents, or RelissueRequest on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The Event header's ID parameter of the NOTIFY request.
IN const CSipPacket& rResponse	The failure response received.

Description

Informs the application that a failure response (<=300) has been received for a NOTIFY request that it sent. Note that this NOTIFY request could be the result of a call to Notify or Terminate.

When the NOTIFY request was sent through the Notify method and the application does not know how to fix the error condition, it should call Terminate.

See Also

ISipClientEventControl (see page 19), ISipNotifierSvc (see page 497)

3.7.1.9.1.3 - ISipNotifierMgr::EvFetched Method

A SUBSCRIBE request to fetch the current state has been received.

C++

```
virtual void EvFetched(IN ISipNotifierSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID of the subscription. It can be an empty string, which means that there is no ID.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

Description

Similar to EvSubscribed (see page 495) but with the "Expires" parameter set to 0. If the application sends a success response for that request (2xx), it must call Terminate. If the authorization process allows to give the state information to that user, it should put it in the content part of that NOTIFY.

If the application rejects the NOTIFY with a final negative response (>=300), it should not call Terminate.

The application should never call Notify on this subscription.

See Also

ISipNotifierSvc (see page 497)

3.7.1.9.1.4 - ISipNotifierMgr::EvInvalidSubscribe Method

A SUBSCRIBE request has been automatically rejected by this service.

C++

```
virtual void EvInvalidSubscribe(IN ISipNotifierSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rSubscribe, IN mxt_result reason) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.
IN mxt_result reason	<ul style="list-style-type: none"> resFE_UNKNOWN_EVENT: The Event header was missing or for an unknown event type. A "489 Bad Event" response has already been sent. resFE_EXPIRED_SUBSCRIPTION: The SUBSCRIBE has been received after a NOTIFY with "subscription-state" "terminated" was sent or after the expiration time. A "481 Subscription does not exist" response has already been sent. resFE_INTERVALL_TOO_BRIEF: The received SUBSCRIBE had a non 0 expiration time smaller than the minimum configured through SetMinimumExpiration. A "423 Interval too brief" response has already been sent.

Description

Informs the application that an invalid SUBSCRIBE has been received. Usually, the application has no action to take other than reporting this invalid packet if wanted. The ISipNotifierMgr (see page 492) that reported the event already took care of sending the appropriate response.

One action that can be taken is to destroy the context if it was created only to handle this SUBSCRIBE as it will not create a dialog.

3.7.1.9.1.5 - ISipNotifierMgr::EvProgress Method

A provisional response (1xx) has been received for a NOTIFY request sent by the ISipNotifierSvc (see page 497).

C++

```
virtual void EvProgress(IN ISipNotifierSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the NOTIFY request was sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must either call CallNextEvent or ClearClientEvents on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID parameter of the Event header of the NOTIFY request.
IN const CSipPacket& rResponse	The provisional response received.

Description

Informs the application that a provisional response has been received for a NOTIFY request that it sent. Note that this NOTIFY request could result of a call to Notify or Terminate.

See Also

ISipClientEventControl (see page 19), ISipSubscriberSvc (see page 595)

3.7.1.9.1.6 - ISipNotifierMgr::EvRefreshed Method

A SUBSCRIBE request to refresh an active subscription has been received.

C++

```
virtual void EvRefreshed(IN ISipNotifierSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN unsigned int uExpirationSec, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID of the subscription. It can be an empty string, which means that there is no ID.
IN unsigned int uExpirationSec	The expiration time suggested in the SUBSCRIBE request. If the request does not suggest an expiration time, it is the default expiration time for that event type.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

Description

This method is called when a valid SUBSCRIBE request is received for a subscription that already exists. Accepting this request with a success response extends the subscription on which the application can call Notify and Terminate.

The application must respond quickly to that request.

See Also

ISipNotifierSvc (see page 497)

3.7.1.9.1.7 - ISipNotifierMgr::EvSubscribed Method

A SUBSCRIBE request for a new subscription has been received.

C++

```
virtual void EvSubscribed(IN ISipNotifierSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN unsigned int uExpirationSec, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the SUBSCRIBE request has been received.

IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID of the subscription. It can be an empty string, which means that there is no ID.
IN unsigned int uExpirationSec	The expiration time suggested in the SUBSCRIBE request. If the request does not suggest an expiration time, it is the default expiration time for that event type.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

Description

This method is called when a valid new SUBSCRIBE request is received. Accepting this request with a success response creates a subscription on which the application can call Notify and Terminate.

The application must respond quickly to that request. If time is needed to authorize a user to access that particular subscription type, a success response should be returned and Notify should be called immediately with the eState parameter set to ePENDING. If the result of the verification is to refuse the subscription, the application would then call Terminate possibly specifying a reason and/or a retry-after value. If the result of the verification is to accept the subscription, the application would then call Notify again but with the eState parameter set to eACTIVE.

Note that if the application knows quickly enough that the subscription cannot be accepted, it can reject it with a failure response.

See Also

ISipNotifierSvc (see page 497)

3.7.1.9.1.8 - ISipNotifierMgr::EvSuccess Method

A success response (2xx) has been received for a NOTIFY request sent by the ISipNotifierSvc (see page 497).

C++

```
virtual void EvSuccess(IN ISipNotifierSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the NOTIFY request was sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must either call CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest is usually called to retry failed requests instead of successful ones.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The Event header's ID parameter of the NOTIFY request.
IN const CSipPacket& rResponse	The success response received.

Description

Informs the application that a success response has been received for a NOTIFY request that it sent. If that NOTIFY request was sent through NOTIFY, the application should consider that the subscription is still active. If the NOTIFY request was sent through a Terminate, the application should consider that this subscription is terminated. It should not call Terminate nor Notify again on that subscription.

See Also

ISipClientEventControl (see page 19), ISipNotifierSvc (see page 497)

3.7.1.9.1.9 - ISipNotifierMgr::EvTerminated Method

A SUBSCRIBE request to terminate an active/pending subscription has been received.

C++

```
virtual void EvTerminated(IN ISipNotifierSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The ID of the subscription. It can be an empty string, which means that there is no ID.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

Description

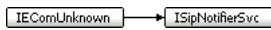
This method is called when a SUBSCRIBE request with an "Expires" header set to 0 is received for an active or pending subscription. If the application accepts this request, it should then call Terminate to send a final NOTIFY.

See Also

ISipNotifierSvc ([see page 497](#))

3.7.1.10 - ISipNotifierSvc Class New in 4.1.4 | Updated behavior in 4.1.4

Class Hierarchy



C++

```
class ISipNotifierSvc : public IEComUnknown;
```

Description

This interface is used to act as a notifier as defined in RFC 3265. It lets the application receive SUBSCRIBE and send NOTIFY messages.

This interface is used to act as a notifier as defined in RFC 3265. It lets the application receive SUBSCRIBE and send NOTIFY for multiple Events. By default, the notifier service does not handle any event types and will reject all subscription requests. Supported events are added through the AddEvent ([see page 499](#)) method. Once event types are thus added they are correctly handled.

This service reports events received to the application through the ISipNotifierMgr ([see page 492](#)) interface.

The ISipNotifierSvc is an ECOM object

The ISipNotifierSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipNotifierSvc

Interface Id: IID_ISipNotifierSvc

A user can query the ISipContext ([see page 23](#)) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext ([see page 23](#)) through the same mean.

Updated behavior: Previously, each event type had its own specific subscriber and notifier services, with its own interface and implementation classes. Thus, for a SIP context to support different event types, many subscriber and/or notifier services had to be individually created and attached to the associated ISipContext ([see page 23](#)). Moreover, new event types required the creation of new child classes deriving from base subscriber or notifier classes, along with supporting infrastructure. This scheme was cumbersome and made supporting new event types unnecessarily complex.

It follows from the foregoing that:

- Supporting many different event types in the same ISipContext ([see page 23](#)) now requires creating a single subscription or notification service, upon which AddEvent ([see page 499](#)) is called many times (once for each event type to support). The service is then added to the context. This previously required many different services to be created and attached.
- Existing user-defined subscription or notification services should be removed from the code base and replaced with the new subscriber or notifier service. Of course, appropriate calls to AddEvent ([see page 499](#)) should also be made. For instance, an "ISipSubscriberProprietaryEventSvc" subscription service should be replaced with a plain ISipSubscriberSvc ([see page 595](#)) upon

which AddEvent (see page 499) ("ProprietaryEvent", ...) is called.

- Adding new user-defined subscription event types is now trivial.

Warning

Old subscriber and notifier services are deprecated and are now removed. By old services we refer to interfaces that were previously named ISipNotifier[EventName]Svc and ISipSubscriber[EventName]Svc.

Location

SipUserAgent/ISipNotifierSvc.h

See Also

ISipNotifierMgr (see page 492)

Methods

Method	Description
◆ A AddEvent (see page 499)	Adds an event type to be supported by this service.
◆ A CreateSubscription (see page 499)	Creates a subscription without receiving a SUBSCRIBE request.
◆ A ExtendImplicitSubscription (see page 500)	Updates the expiration time for an implicit subscription.
◆ A GetCurrentSubscriptions (see page 500)	Gets the currently active subscriptions.
◆ A Notify (see page 501)	Sends a NOTIFY request for the specified subscription.
◆ A SetManager (see page 502)	Configures the manager of this service.
◆ A SetMinimumExpiration (see page 502)	Sets the minimum acceptable expiration time.
◆ A Terminate (see page 502)	Sends a final NOTIFY for that subscription.

Legend

◆ A	Method
◆ A	abstract

Enumerations

Enumeration	Description
EReason (see page 503)	
EState (see page 504)	
EType (see page 504)	

Structs

Struct	Description
SSubscriptionInfo (see page 498)	

3.7.1.10.1 - Structs

3.7.1.10.1.1 - ISipNotifierSvc::SSubscriptionInfo Struct

```
struct SSubscriptionInfo {
    CString m_strEvent;
    CString m_strId;
};
```

Description

Information that uniquely identify a subscription.

Members

Members	Description
CString m_strEvent;	The event type, e.g., "message-summary".
CString m_strId;	The "id" parameter of the subscription.

3.7.1.10.2 - Methods

3.7.1.10.2.1 - ISipNotifierSvc::AddEvent Method

Adds an event type to be supported by this service.

C++

```
virtual mxt_result AddEvent(IN const CString& rstrEvent, IN unsigned int uDefaultExpiration) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	Event type to support, e.g., "message-summary".
IN unsigned int uDefaultExpiration	Default expiration for this event type, in seconds.

Returns

resS_OK: Success.

Description

Adds an event type to be supported by this service.

3.7.1.10.2.2 - ISipNotifierSvc::CreateSubscription Method

Creates a subscription without receiving a SUBSCRIBE request.

C++

```
virtual mxt_result CreateSubscription(IN const CString& rstrEvent, IN const CString& rstrId, IN unsigned int uExpirationSec, IN EType eType) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration for that event type is used. This argument is ignored when eType is eIMPLICIT_NOEXPIRES.
IN EType eType	Can be eEXPLICIT, which means that the subscription has been established by SIP signalling but not via a SUBSCRIBE request. When the subscription is established by other means than SIP, the value of this parameter should be eIMPLICIT or eIMPLICIT_NOEXPIRES to ignore uExpirationSec.
strId	The "id" parameter in the Event header field of the SUBSCRIBE received for that subscription. When using the empty string, the "id" parameter must not be in the SUBSCRIBE message to match that subscription.

Returns

- resFE_INVALID_STATE: The manager has not been set.
- resFE_INVALID_ARGUMENT: The id is already used by another subscription in this object or the event is unknown.
- resS_OK: The subscription has been successfully created and ready to send NOTIFY requests.

Description

Creates a subscription without receiving a SUBSCRIBE request. Set the eType parameter to eEXPLICIT when the subscription was created by some other SIP means (eg. Receiving a REFER request). Set the eType parameter to eIMPLICIT when the subscription is created by some other non SIP means like a common agreement. Set the eType parameter to eIMPLICIT_NOEXPIRES when you want an implicit subscription that will never expire. The value of this parameter will dictate what is to be done when the subscription is terminated or refreshed. When the subscription is explicit, calling Refresh will continue the implicit subscription. When the subscription is explicit, it must be refreshed by the remote peer sending a SUBSCRIBE request in order to stay active.

When establishing an explicit subscription, because of the reception of a non-SUBSCRIBE request, the application should wait to send a success response to that request to call CreateSubscription. It is then able to send a NOTIFY immediately.

Otherwise, the subscription created by this method works exactly the same way as the one created by sending a success response to a received SUBSCRIBE request.

See Also

Terminate (see page 502)

3.7.1.10.2.3 - **ISipNotifierSvc::ExtendImplicitSubscription** Method

Updates the expiration time for an implicit subscription.

C++

```
virtual mxt_result ExtendImplicitSubscription(IN const CString& rstrEvent, IN const CString& rstrId, IN unsigned int uExpirationSec) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration for that event type is used.
strId	The "id" parameter in the Event header field of the SUBSCRIBE received for that subscription. When using the empty string, the "id" parameter must not be in the SUBSCRIBE message to match that subscription.

Returns

- resFE_INVALID_ARGUMENT: The id does not correspond to any subscription in this object or the event is unknown.
- resFE_INVALID_STATE: The id corresponds to an explicit subscription.
- resS_OK: The implicit subscription expiration is updated.

Description

Updates the expiration time for an implicit subscription. It lets the application continue to call Notify (see page 501) for an implicit subscription after the original amount of time that was configured through CreateSubscription (see page 499).

This method should be called when EvExpired is called for an implicit subscription and the application wants to continue the subscription.

See Also

ISipNotifierMgr::EvExpired (see page 492), CreateSubscription (see page 499), AddEvent (see page 499)

3.7.1.10.2.4 - **ISipNotifierSvc::GetCurrentSubscriptions** Method

Gets the currently active subscriptions.

C++

```
virtual void GetCurrentSubscriptions(OUT CVector<SSubscriptionInfo>& rvecstSubscriptions) = 0;
```

Parameters

Parameters	Description
OUT CVector<SSubscriptionInfo>& rvecstSubscriptions	A vector to hold the currently active subscriptions.

Description

Returns in a vector the information about each currently active subscription.

3.7.1.10.2.5 - ISipNotifierSvc::Notify Method

Sends a NOTIFY request for the specified subscription.

C++

```
virtual mxt_result Notify(IN const CString& rstrEvent, IN const CString& strId, IN EState eSubscriptionState,
IN unsigned int uExpirationSec, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO
CSipMessageBody* pMessageBody, IN TO CGenParamList* pCustomParameters, OUT ISipClientTransaction*&
rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN EState eSubscriptionState	The state to put in that subscription. Values can be ePENDING, which usually means it is the first NOTIFY and that the application is verifying if the user has the right to subscribe to this resource. A "Subscription-State" header with the value "pending" is added to the NOTIFY request. Otherwise, the value is eACTIVE, which means that the subscription is authorized and active. A "Subscription-State" header with the value "active" is added to the NOTIFY request.
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the current remaining expiration for that subscription is used. This value is put in the "Subscription-State" header's "expires" parameter of the NOTIFY request sent.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the sent NOTIFY request. It can be NULL, which means that no message-body is added to the request. Ownership of this parameter is TAKEN.
IN TO CGenParamList* pCustomParameters	A pointer to the parameters added to the Event header of the sent NOTIFY request. It can be NULL, which means that no parameter is added to the request. Ownership of this parameter is TAKEN. This list should not contain the "id" parameter since it will be added from the strId parameter. If the "id" parameter is present, its value will be replaced with strId.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
strId	The "id" parameter of the subscription. It is put in Event header of the NOTIFY request sent.

Returns

- **resFE_INVALID_ARGUMENT:** The id does not correspond to any subscription in this object, the expiration time is higher than the remaining time for that subscription or the event is unknown.
- **resFE_INVALID_STATE:** The id corresponds to a subscription that is expired. Use Terminate (see page 502) instead to send a final NOTIFY. Note that this is the case when receiving a SUBSCRIBE with a "Expires" header value of 0 OR the attached ISipUserAgentSvc (see page 639) is incorrectly configured. For instance, the contact list could be empty.
- **resFE_FAIL:** The NOTIFY request could not be sent, for instance because there is an outstanding NOTIFY which didn't yet receive a response.
- **resFE_SIPCORE_PACKET_BLOCKED:** One of the service synchronously blocked the packet. No additional event will be reported.
- **resS_OK:** The NOTIFY request has been successfully sent.

Description

Sends a NOTIFY request for an active or pending subscription. The application may add the proper content for that event type.

When the application lowers the expiration time by using a non NULL value in the uExpirationSec parameter, the expiration time is lowered if the NOTIFY request receives a success response.

See Also

Terminate (see page 502), AddEvent (see page 499)

3.7.1.10.2.6 - ISipNotifierSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipNotifierMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipNotifierMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.10.2.7 - ISipNotifierSvc::SetMinimumExpiration Method

Sets the minimum acceptable expiration time.

C++

```
virtual mxt_result SetMinimumExpiration(IN const CString& rstrEvent, IN unsigned int uMinExpirationSec) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	Event type, e.g., "message-summary".
IN unsigned int uMinExpirationSec	The minimum acceptable expiration time. When 0, it means that the expiration time can be as low as the subscriber wants. The application should not set this value higher than 1 hour (3600 seconds) in order to be compliant with RFC 3265.

Returns

- resFE_INVALID_ARGUMENT: The event type is unknown.

Description

Sets the minimum acceptable expiration time for a subscription of the given event type. When receiving a SUBSCRIBE request with this event type and with a value lower than this value but higher than 0, a "423 Interval too small" error is automatically sent.

3.7.1.10.2.8 - ISipNotifierSvc::Terminate Method

Sends a final NOTIFY for that subscription.

C++

```
virtual mxt_result Terminate(IN const CString& rstrEvent, IN const CString& rstrId, IN EReason eReason, IN
unsigned int uRetryAfterSec, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO
CSipMessageBody* pMessageBody, IN TO CGenParamList* pCustomParameters, OUT ISipClientTransaction*&
```

```
rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN EReason eReason	The "reason" parameter to put in the "Subscription-State" header. When the value is eNO_REASON, the parameter is not put in the "Subscription-State" header.
IN unsigned int uRetryAfterSec	The number of seconds to put in the "retry-after" parameter of the "Subscription-State" header. When the value is 0, no "retry-after" parameter is put. Note that RFC 3265 defines semantics for this parameter only if the eReason parameter has the value eNO_REASON, ePROBATION, or eGIVEUP.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the sent NOTIFY request. It can be NULL, which means that no message-body is added to the request. Ownership of this parameter is TAKEN.
IN TO CGenParamList* pCustomParameters	A pointer to the parameters added to the Event header of the sent NOTIFY request. It can be NULL, which means that no parameter is added to the request. Ownership of this parameter is TAKEN. This list should not contain the "id" parameter since it will be added from the strId parameter. If "id" parameter is present, its value will be replaced with strId.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
strId	The "id" parameter of the subscription. It is put in Event header of the NOTIFY request sent.

Returns

- resFE_INVALID_ARGUMENT: The id does not correspond to any subscription in this object or the event is unknown.
- resFE_FAIL: The NOTIFY request could not be sent, for instance because there is an outstanding NOTIFY which didn't yet receive a response.
- resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
- resS_OK: The NOTIFY request has been successfully sent.

Description

Sends a NOTIFY with the "Subscription-State" header set to "terminated". The application may add the proper content for that event type.

This method should be called when the event EvExpired is called by the ISipNotifierMgr (see page 492).

It should also be called when accepting a SUBSCRIBE received through the EvFetched event or the EvTerminated event of the ISipNotifierMgr (see page 492).

See Also

Notify (see page 501), EvExpired, EvFetched, EvTerminated, AddEvent (see page 499)

3.7.1.10.3 - Enumerations

3.7.1.10.3.1 - ISipNotifierSvc::EReason Enumeration

```
enum EReason {
    eDEACTIVATED,
    ePROBATION,
    eREJECTED,
```

```

    eTIMEOUT,
    eGIVEUP,
    eNO_RESOURCE,
    eNO_REASON
};

ePROBATION
eREJECTED
eTIMEOUT
eGIVEUP
eNO_RESOURCE

```

Description

Indicate why the subscription is terminating. See Terminate (see page 502).

Members

Members	Description
eDEACTIVATED	Reason to terminate the subscription.
eNO_REASON	No reason parameter will be added to the "Subscription-State" header.

3.7.1.10.3.2 - ISipNotifierSvc::EState Enumeration

```

enum EState {
    eACTIVE,
    ePENDING
};

```

Description

Indicates the state of the subscription. See Notify (see page 501) for more information.

Members

Members	Description
eACTIVE	The subscription is active.
ePENDING	The subscription is pending.

3.7.1.10.3.3 - ISipNotifierSvc::EType Enumeration

```

enum EType {
    eIMPLICIT,
    eEXPLICIT,
    eIMPLICIT_NOEXPIRES
};

```

Description

Indicate how the subscription is created. See CreateSubscription (see page 499) for more information.

Members

Members	Description
eIMPLICIT	The subscription is established by other means than SIP
eEXPLICIT	The subscription has been established by SIP signalling but not via a SUBSCRIBE request
eIMPLICIT_NOEXPIRES	Same as eIMPLICIT but the expiration time will be ignored

3.7.1.11 - ISipOptionTagsMgr Class **New in 4.1.8**

Class Hierarchy

```
ISipOptionTagsMgr
```

C++

```
class ISipOptionTagsMgr;
```

Description

The Option Tags manager is the interface through which the Option Tags service reports events to the application. The Option Tags

service informs the application through this interface of incoming requests with `Require` header values that does not match the application's supported capabilities.

Location

`SipUserAgent/ISipOptionTagsMgr.h`

See Also

`ISipOptionTagsMgr`

Methods

Method	Description
• A <code>EvUnsupportedTag</code> (see page 505)	Reports that a SIP request has been automatically rejected.

Legend

•	Method
A	abstract

3.7.1.11.1 - Methods

3.7.1.11.1.1 - `ISipOptionTagsMgr::EvUnsupportedTag` Method

Reports that a SIP request has been automatically rejected.

C++

```
virtual void EvUnsupportedTag(IN ISipOptionTagsSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipOptionTagsSvc* pSvc	The <code>ISipOptionTagsSvc</code> (see page 96) that refused the request.
IN mxt_opaque opqApplicationData	The application opaque data.
IN const CSipPacket& rRequest	The rejected SIP request.

Description

Informs the manager when the `ISipOptionTagsSvc` (see page 96) automatically refuses an incoming request with a 420 response. The application can clear/release the context upon reception of this event.

3.7.1.12 - `ISipPrivacyMgr` Class

Class Hierarchy

<code>ISipPrivacyMgr</code>

C++

```
class ISipPrivacyMgr;
```

Description

`ISipPrivacyMgr` is the interface through which the `ISipPrivacySvc` (see page 507) reports events to the application.

`ISipPrivacySvc` (see page 507) reports an event when a request is received from an IP address that differs from the configured privacy service provider.

Location

`SipUserAgent/ISipPrivacyMgr.h`

See Also

`ISipPrivacySvc` (see page 507)

Methods

Method	Description
• A EvPrivacyDnsResolutionCompleted (see page 506)	The privacy list has been resolved.
• A EvUncertifiedPrivacy (see page 506)	A request has been received from another host than the configured privacy service provider.

Legend

• A	Method
A	abstract

3.7.1.12.1 - Methods

3.7.1.12.1.1 - ISipPrivacyMgr::EvPrivacyDnsResolutionCompleted Method

The privacy list has been resolved.

C++

```
virtual void EvPrivacyDnsResolutionCompleted(IN ISipPrivacySvc* pSvc, IN bool bSharedTrustedPrivacy) = 0;
```

Parameters

Parameters	Description
IN ISipPrivacySvc* pSvc	The service managing privacy.
IN bool bSharedTrustedPrivacy	<ul style="list-style-type: none"> • true: this event is for a SetSharedPrivacyService call. • false: this event is for a SetInstancePrivacyService call.

Description

This method is called when the DNS resolution of the trusted hosts URI list is completed. It will be called for either the shared or instance trusted hosts as indicated by the bSharedTrustedPrivacy parameter.

Before this event is reported to the application, all other events of this manager SHOULD be considered unreliable. Once this event has been reported at least once for either the shared or instance trusted proxy list, all events are reliable.

This means that when the shared trusted hosts are set and the EvPrivacyDnsResolutionCompleted event has been reported once, changing the instance trusted host will NOT render the events unreliable. Those events could be reported while the DNS resolution for that new list is in progress and they will be based on the shared list until the DNS resolution is completed.

The events reliability is based on the fact that there is at least one trusted host IP address list set. It does not matter if it is the shared or the instance one. As long as there is a trusted host IP address list set, the events are reliable according to that list.

The rule of thumb is: once the EvPrivacyDnsResolutionCompleted event has been reported, the service's events are reliable.

3.7.1.12.1.2 - ISipPrivacyMgr::EvUncertifiedPrivacy Method

A request has been received from another host than the configured privacy service provider.

C++

```
virtual void EvUncertifiedPrivacy(IN ISipPrivacySvc* pSvc, IN const CSipPacket& rRequest, INOUT mxt_opaque& rOpqApplicationData) = 0;
```

Parameters

Parameters	Description
IN ISipPrivacySvc* pSvc	The service managing privacy.
IN const CSipPacket& rRequest	The request that was received from another host than the configured privacy service.

INOUT mxt_opaque& rOpqApplicationData	Application data opaque to the service. This parameter is an INOUT parameter. It is used to correlate the events reported by multiple services for a unique received request. If the application has already received an event for that request through another manager interface, rApplicationData equals the value stored in it by the application. Otherwise, if it is the first event reported for this received request, rApplicationData is set to 0. The value of rApplicationData when EvUncertifiedPrivacy returns is accessible through the GetOpaque() method of the ISipServerEventControl (see page 75) interface that accompanies the request when the owner service issues its event. This opaque data should be used to store the state indicating that the privacy may be compromised by accepting this request and being able to act accordingly when the session manager receives the event containing the ISipServerEventControl (see page 75). Note that this event cannot be processed asynchronously since the opaque application data is passed by value to the ISipServerEventControl (see page 75) interface.
---------------------------------------	---

Description

This event means that a request has been received from another IP address than the resolved configured privacy service provider. This means that by accepting the request, the user agent could send identity-sensitive informations on the network.

When receiving this event, the application should send a final negative response to the request. The application should redirect the request (send a 3xx response) to its 'anonymous callback' URI if it knows it.

The address of the entity from which the packet was received can be retrieved by rRequest.GetPeerAddr().

See Also

ISipPrivacySvc (see page 507), ISipPrivacySvc::SetSharedPrivacyService (see page 511),
ISipPrivacySvc::SetInstancePrivacyService (see page 510), CSipPacket::GetPeerAddr (see page 453)

3.7.1.13 - ISipPrivacySvc Class

Class Hierarchy



C++

```
class ISipPrivacySvc : public IEComUnknown;
```

Description

This service implements user-level privacy functions as per RFC 3323.

The service removes all headers that may contain or give a hint of private information. These headers are Call-Info, Reply-To, User-Agent, Organization, Server, Subject, In-Reply-To, and Warning.

It is important to know that other headers may contain private information. Some of these headers are From, Call-ID, and Referred-By. It is the application's responsibility to properly set the value of these headers to be sure they do not contain private information. Some of these headers are set through ISipUserAgentSvc (see page 639). Note that the default Call-ID value generated by the ISipUserAgentSvc (see page 639) does not contain any private information, so it can be kept as is to guarantee its privacy.

When using this service, the application should be cautious when adding headers. It must make sure that it does not include private information in added headers.

Also, other elements of a packet must give informations on how the current dialog should be routed. These elements are the Contact header, the Via header, and the session information in the SDP. These must contain valid informations and cannot be anonymized by the local UA. To transform these elements, the user agent must use an intermediary. This intermediary is a privacy service provider (also named a privacy service). To anonymize the headers, the user agent must request the 'header' privacy type. To anonymize the SDP informations, the user agent must request the 'session' privacy type. Note that when 'session' privacy type is requested, the SDP payload MUST NOT be encrypted.

This service allows to set a privacy service provider that will provide further privacy if requested. The privacy type handled by the privacy service provider is configurable via the SetPrivacyType (see page 511) method. By default, the 'header' and 'id' privacy types are requested. An application should couple the use of this service with the use of a privacy service provider.

Note that the application must route the request to the privacy service provider. This should be done by setting a pre-loaded route to the privacy service provider. The top-most Route header of the pre-loaded route must contain the privacy service provider URI. Also, if a user agent is sending a request directly to a privacy service, it SHOULD include a Proxy-Require header containing the 'privacy' option-tag, especially when the 'critical' priv-value is present in the Privacy header. Configuring the Proxy-Require header can be done by using the AddProxyRequire (see page 509) method. The header is added by default.

RFC 3323 states that to certify privacy while receiving a request, the user agent can "procure an 'anonymous callback' URI from the third-party service and [...] distribute this as an address-of-record. A privacy service provider might offer these anonymous callback URIs to users in the same way that an ordinary SIP service provider grants addresses-of-record. The user would then register their normal address-of-record as a contact address with the third-party service." This means that requests should only come from the privacy service provider.

RFC 3323 also gives other alternatives. One of them is that the "user agent could send REGISTER requests through a privacy service with a request for 'user' level privacy. This will allow the privacy service to insert anonymous Contact header URIs. Requests sent to the user's conventional address-of-record would then reach the user's devices without revealing any usable contact addresses."

In any way, to have privacy guaranteed, requests must be received from the privacy service provider. The service warns the application through the `ISipPrivacyMgr::EvUncertifiedPrivacy` (see page 506) event if a request has been received from a source other than the configured privacy service provider. The application can then choose to reject the request, accept it with user approval, or redirect it to its 'anonymous callback' URI.

Note that private information is transmitted in transactions between the user agent and the privacy service provider, so RFC 3323 highly recommends "that user agents use network or transport layer security, such as TLS, when contacting a privacy service."

This service generates events to the application through the `ISipPrivacyMgr` (see page 505) interface.

IMPORTANT: this service must be the LAST added to the context. Doing otherwise may lead to privacy issues.

The `ISipPrivacySvc` is an ECOM object

The `ISipPrivacySvc` is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: `CLSID_CSipPrivacySvc`

Interface Id: `IID_ISipPrivacySvc`

A user can query the `ISipContext` (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the `ISipContext` (see page 23) through the same mean.

Location

`SipUserAgent/ISipPrivacySvc.h`

See Also

`ISipPrivacyMgr` (see page 505), `ISipUserAgentSvc::SetPreloadedRoute` (see page 653)

Methods

Method	Description
• A <code>AddProxyRequire</code> (see page 509)	Enables or disables the usage of the <code>Proxy-Require</code> header containing the 'privacy' option-tag.
• A <code>EnableHeaderRemoval</code> (see page 509)	Enables the service's removal of certain headers.
• A <code>SetInstancePrivacyService</code> (see page 510)	Sets the address of the privacy service provider used by this instance of this service.
• A <code>SetManager</code> (see page 510)	Configures the manager associated with this instance.
• A <code>SetPrivacyType</code> (see page 511)	Sets the privacy types requested to the privacy service provider.
• A <code>SetSharedPrivacyService</code> (see page 511)	Sets the address of the privacy service provider used by all instances of this service.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
<code>EPriValue</code> (see page 512)	Requested privacy level.

3.7.1.13.1 - Methods

3.7.1.13.1.1 - ISipPrivacySvc::AddProxyRequire Method

Enables or disables the usage of the Proxy-Require header containing the 'privacy' option-tag.

C++

```
virtual mxt_result AddProxyRequire(IN bool bAddHeader) = 0;
```

Parameters

Parameters	Description
rPrivacyService	true if a Proxy-Require header with the 'privacy' option-tag must be added to outgoing requests.

Returns

resFE_INVALID_STATE: This service has no manager. Maybe it was not set or Clear was called on the context.

resS_OK: The configuration has been set.

Description

This method permits to configure whether a Proxy-Require with the 'privacy' option-tag must be added to outgoing requests.

RFC 3323 states that the header SHOULD be added especially when the 'critical' privacy value is present in the Privacy header.

The header is added by default.

This configuration only affects the instance upon which it is called.

See Also

SetPrivacyType (see page 511)

3.7.1.13.1.2 - ISipPrivacySvc::EnableHeaderRemoval Method

Enables the service's removal of certain headers.

C++

```
virtual void EnableHeaderRemoval(IN bool bPrivacy) = 0;
```

Parameters

Parameters	Description
IN bool bPrivacy	True to activate service privacy, false to deactivate it.

Description

This method is used to activate or deactivate the service's innate behaviour of stripping specific SIP headers automatically:

- Call info;
- Organization;
- Reply to;
- User agent;
- In reply to (request only);
- Subject (request only);
- Server (response only);
- Warning (response only).

In certain circumstances, when an application wants to downgrade the privacy level requested from the server, it may want to keep these headers. It must call this method and set bPrivacy to false.

See Also

SetPrivacyType (see page 511)

3.7.1.13.1.3 - ISipPrivacySvc::SetInstancePrivacyService Method

Sets the address of the privacy service provider used by this instance of this service.

C++

```
virtual mxt_result SetInstancePrivacyService(IN const CSipUri& rPrivacyService) = 0;
```

Parameters

Parameters	Description
IN const CSipUri& rPrivacyService	The SIP-URI / SIPS-URI of the server that will provide privacy to this context.

Returns

resFE_INVALID_STATE: This service has no manager. Maybe it was not set or Clear was called on the context.

resFE_INVALID_STATE: The privacy service provider list is currently used. The main cause would be that resolving is currently processed for the service list.

resS_OK: Resolving has begun for the privacy service provider URI passed to the method.

Description

Sets the address of the privacy service provider. It is through this privacy service provider that outgoing requests and responses may be further anonymized.

This method overrides, for this instance only, all previous and future configuration done through SetSharedPrivacyService (see page 511). That is, this service always uses its instance configuration first, and if none is set, then it uses the shared configuration.

If the system only uses a single privacy service provider, you should use SetSharedPrivacyService (see page 511) once in the system as DNS resolving for the server is done just once. If the system simultaneously uses more than one privacy service provider, you have to use SetInstancePrivacyService each time this service is attached to a SIP Context.

This method resolves the SIP-URI / SIPS-URI to one or more IP addresses.

If requests are received from a server other than the configured privacy service, EvUncertifiedPrivacy is reported to the application.

The application SHOULD set a pre-loaded route, with the first route pointing to the privacy service provider. This guarantees that requests are always sent first to the privacy service server, which can then further anonymize the requests and responses. Doing otherwise would compromise privacy.

See Also

ISipPrivacyMgr::EvUncertifiedPrivacy (see page 506), SetSharedPrivacyService (see page 511)

3.7.1.13.1.4 - ISipPrivacySvc::SetManager Method

Configures the manager associated with this instance.

C++

```
virtual mxt_result SetManager(IN ISipPrivacyMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipPrivacyMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT: pMgr is NULL. The manager is not changed in this case.

resS_OK: Otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.13.1.5 - ISipPrivacySvc::SetPrivacyType Method

Sets the privacy types requested to the privacy service provider.

C++

```
virtual mxt_result SetPrivacyType( IN unsigned int uPrivacyValues ) = 0;
```

Parameters

Parameters	Description
IN unsigned int uPrivacyValues	The privacy header values that must be requested to the privacy service provider. These values are added to a Privacy header into outgoing packets. Many option-tags can be set by using the ' ' operator. Since no option-tag must be added with the 'none' privacy type, setting eNONE with other privacy type returns an error. Setting only eCRITICAL also fails.

Returns

resFE_INVALID_STATE: This service has no manager. Maybe it was not set or Clear was called on the context.

resFE_INVALID_ARGUMENT: uPrivacyValues is invalid. Either eNONE is present with other option-tags, eCRITICAL is present alone, or the values were not taken from EPrivValue (see page 512).

resS_OK: otherwise.

Description

This method permits to change the privacy header values that must be added to the Privacy header of outgoing packets (requests and responses).

By default, the 'header' and 'id' privacy types are requested.

RFC 3323 states that no other option-tag must be present with the "none" option-tag. To be compliant with that, when a packet already contains a Privacy header and this service updates it, the service replaces all the privacy types of the header by 'none' if the configured privacy type is eNONE. For example, "Privacy: abc" is replaced by "Privacy: none" when the configured privacy type is eNONE.

Also, if the configured privacy type is not eNONE, the privacy type(s) is appended to the existing one. For example, "Privacy: abc" becomes "Privacy: abc;header;session" with eHEADER | eSESSION configured as privacy types.

Note that the Privacy header is not added to CANCEL requests.

This configuration only affects the instance upon that it is called.

See Also

ISipPrivacySvc::EPrivValue (see page 512)

Example

The following shows how this method should be called.

```
// Enable "header", "session" and "critical" option-tags
pSvc->SetPrivacyType(eHEADER | eSESSION | eCRITICAL);
```

3.7.1.13.1.6 - ISipPrivacySvc::SetSharedPrivacyService Method

Sets the address of the privacy service provider used by all instances of this service.

C++

```
virtual mxt_result SetSharedPrivacyService( IN const CSipUri& rPrivacyService ) = 0;
```

Parameters

Parameters	Description
IN const CSipUri& rPrivacyService	The SIP-URI / SIPS-URI of the server that provides privacy to the local UA.

Returns

resFE_INVALID_STATE: This service has no manager. Maybe it was not set or Clear was called on the context.

resFE_INVALID_STATE: This service already has its own instance list so modifying the shared list would not affect the current service behaviour. Updating was not done.

resFE_INVALID_STATE: The privacy service provider list is currently used. The main cause would be that resolving is currently processed for the service list.

resS_OK: Resolving has begun for the privacy service provider URI passed to the method.

Description

Sets the address of the privacy service provider. It is through this privacy service provider that outgoing requests and responses may be further anonymized.

This method configures all existing and future instances of this service with rPrivacyService, but does not affect the instances that had their privacy service provider configured through SetInstancePrivacyService (see page 510).

If the system only uses a single privacy service provider, you should use SetSharedPrivacyService once in the system as DNS resolving for the server is done just once. If the system simultaneously uses more than one privacy service provider, you have to use SetInstancePrivacyService (see page 510) each time this service is attached to a SIP Context.

This method resolves the SIP-URI / SIPS-URI to one or more IP addresses.

If requests are received from a server other than the privacy service, EvUncertifiedPrivacy is reported to the application.

A pre-loaded route SHOULD be set by the application, with the first route pointing to the privacy service provider. This is to guarantee that requests are always sent first to the privacy service server, which can then further anonymize the requests and responses. Doing otherwise would compromise privacy.

See Also

ISipPrivacyMgr::EvUncertifiedPrivacy (see page 506), SetInstancePrivacyService (see page 510)

3.7.1.13.2 - Enumerations

3.7.1.13.2.1 - ISipPrivacySvc::EPriValue Enumeration

Requested privacy level.

C++

```
enum EPriValue {
    eHEADER = 1,
    eSESSION = eHEADER*2,
    eUSER = eSESSION*2,
    eNONE = eUSER*2,
    eCRITICAL = eNONE*2,
    eID = eCRITICAL*2,
    eINVALID = eID*2
};
```

Description

Bitset describing the privacy level that can be requested. To get a complete description of each level, please refer to RFC 3323 sections 4.2 and 5, or refer to RFC 3325.

Members

Members	Description
eHEADER = 1	1
eSESSION = eHEADER*2	2
eUSER = eSESSION*2	4
eNONE = eUSER*2	8
eCRITICAL = eNONE*2	16
eID = eCRITICAL*2	32
eINVALID = eID*2	64

3.7.1.14 - ISipPublishMgr Class

Class Hierarchy

```
ISipPublishMgr
```

C++

```
class ISipPublishMgr;
```

Description

The Publish Manager is the interface through which the publish service talks to the application. The ISipPublishSvc (see page 516) informs the application through this interface of the progress of the PUBLISH requests it has sent.

Location

SipUserAgent/ISipPublishMgr.h

See Also

ISipPublishSvc (see page 516)

Methods

Method	Description
◆ A EvConditionalRequestFailed (see page 513)	Notifies the application that a 412 final negative response has been received.
◆ A EvExpired (see page 514)	Notifies the application that the publication has expired.
◆ A EvExpiresIntervalTooBrief (see page 514)	Notifies the application that a 423 final negative response has been received.
◆ A EvExpiring (see page 514)	Notifies the application that the publication is about to expire.
◆ A EvFailure (see page 515)	A failure response (>= 300) to a PUBLISH request has been received.
◆ A EvProgress (see page 515)	A provisional response (1xx) to a PUBLISH request is received.
◆ A EvProtocolError (see page 515)	Notifies the application that an invalid 200 response has been received for a publication.
◆ A EvSuccess (see page 516)	A success response (2xx) to a PUBLISH request has been received.

Legend

◆ A	Method
A	abstract

3.7.1.14.1 - Methods

3.7.1.14.1.1 - ISipPublishMgr::EvConditionalRequestFailed Method

Notifies the application that a 412 final negative response has been received.

C++

```
virtual void EvConditionalRequestFailed(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN
unsigned int uExpiresS, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN unsigned int uExpiresS	The value of the Expires header.
IN const CSipPacket& rResponse	The actual response.

Description

Notifies the publish manager that a 412 final negative response has been received, indicating that the precondition given for the request has failed. This means there is no matching event state at the ESC to be refreshed, modified, or removed. The EPA SHOULD perform an initial publication. Before reporting this event, the publish service clears its entity-tag.

See Also

[ISipPublishSvc::Publish](#) (see page 518), [ISipClientEventControl::ReissueRequest](#) (see page 20)

3.7.1.14.1.2 - ISipPublishMgr::EvExpired Method

Notifies the application that the publication has expired.

C++

```
virtual void EvExpired(IN ISipPublishSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The ISipPublishSvc (see page 516) that is managing the publication.

Description

Notifies the publish manager that the publication managed by the publish service has expired. The manager should consider that this entity-tag is no longer published to the server. Usually, the manager should not let that event occur. If it wants to remove a publication, it should not wait until its expiration, but rather explicitly remove it with the Remove method of the publish service interface.

If the manager wants to keep the publication to the server, it should publish it again by calling the Publish method of the publish service. This event should only occur when the manager was unable to refresh or remove its publication due to some remote server error and when the publication was received with a smaller expiration time than the threshold of the publish service.

See Also

[ISipPublishSvc](#) (see page 516), [EvExpiring](#) (see page 514)

3.7.1.14.1.3 - ISipPublishMgr::EvExpiresIntervalTooBrief Method

Notifies the application that a 423 final negative response has been received.

C++

```
virtual void EvExpiresIntervalTooBrief(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN
unsigned int uExpiresS, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN unsigned int uExpiresS	The minimal expiration time accepted by the server.
IN const CSipPacket& rResponse	The actual response.

Description

Notifies the publish manager that a 423 response has been received, meaning that the requested publication expiration time is too brief for the server. The application must therefore issue another Publish request with a higher Expires value.

When reporting this event, the ISipPublishSvc (see page 516) will have adjusted the application requested expiration time to be equal to the minimal expiration time accepted by the server. The stack user receiving this event has the choice to abandon the publication, try it again with the minimal expiration time by calling ReissueRequest on pClientEventCtrl, or call again Publish with the expiration time it sees fit.

See Also

[ISipPublishSvc::Publish](#) (see page 518), [ISipClientEventControl::ReissueRequest](#) (see page 20)

3.7.1.14.1.4 - ISipPublishMgr::EvExpiring Method

Notifies the application that the publication is about to expire.

C++

```
virtual void EvExpiring(IN ISipPublishSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The ISipPublishSvc (see page 516) that is managing the contacts.

Description

Notifies the publish manager that the publication is about to expire. If the manager wants to keep this publication active on the server, it should refresh it by calling the Refresh method of the publish service. If the manager fails to do so, EvExpired (see page 514) is issued later, the amount of time being dependent on ISipPublishSvc::SetExpiringThreshold (see page 521)'s configuration.

See Also

ISipPublishSvc (see page 516), EvExpired (see page 514), Refresh

3.7.1.14.1.5 - ISipPublishMgr::EvFailure Method

A failure response (>= 300) to a PUBLISH request has been received.

C++

```
virtual void EvFailure(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The ISipPublishSvc (see page 516) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The response packet.

Description

Notifies the publish manager that a failure response has been received for the last outgoing PUBLISH request sent through the publish service.

After this failure response, the publish service acts as if the last PUBLISH request was not sent.

3.7.1.14.1.6 - ISipPublishMgr::EvProgress Method

A provisional response (1xx) to a PUBLISH request is received.

C++

```
virtual void EvProgress(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The ISipPublishSvc (see page 516) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The response packet.

Description

Notifies the publish manager that a provisional response has been received for the last outgoing PUBLISH request sent through the publish service.

The state of the publish service does not change after the reception of this provisional response: it still waits for a final response.

3.7.1.14.1.7 - ISipPublishMgr::EvProtocolError Method

Notifies the application that an invalid 200 response has been received for a publication.

C++

```
virtual void EvProtocolError(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The actual response.

Description

Notifies the publish manager that an invalid 2xx response has been received. This response either contains no SIP-ETag header or multiple headers of such type. According to the SIP specification, it is a must to have one and only one SIP-ETag header per 2xx response.

The publish service clears its entity-tag and stops any timer (related to EvExpiring (see page 514) and EvExpired (see page 514)) it holds before reporting this event.

See Also

ISipPublishSvc::Publish (see page 518), ISipClientEventControl::RelssueRequest (see page 20)

3.7.1.14.1.8 - ISipPublishMgr::EvSuccess Method

A success response (2xx) to a PUBLISH request has been received.

C++

```
virtual void EvSuccess(IN ISipPublishSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipPublishSvc* pSvc	The ISipPublishSvc (see page 516) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rResponse	The response packet.

Description

Notifies the publish manager that a success response has been received for the last outgoing PUBLISH request sent through the publish service.

After this success response, the publish service keeps track of the entity-tag and warns the manager when the publication is about to expire and when it has expired.

3.7.1.15 - ISipPublishSvc Class**Class Hierarchy****C++**

```
class ISipPublishSvc : public IEComUnknown;
```

Description

This Publish (see page 518) service implements RFC 3903. It offers the interface to manage a publication from an Event Publication Agent (EPA) to an event state compositor (ESC).

It monitors the expiration of any publication and notifies the manager via the ISipPublishMgr::EvExpiring (see page 514) event a few seconds before expiring. It also reports the ISipPublishMgr::EvExpired (see page 514) event when the publication has expired. The number of seconds between ISipPublishMgr::EvExpiring (see page 514) and the publication expiration can be configured through the SetExpiringThreshold (see page 521) method.

This service does not initiate publication refreshes, modifications, or removals on its own. This must always be initiated by the user of this service. The service is not responsible to manage message bodies, it is the application's responsibility.

The application has the flexibility to get and set the entity-tag held by this service. This allows the application to place any entity-tag it wants in a PUBLISH request. For instance, this is useful after the application has rebooted and wants to issue a PUBLISH request to an already established publication with a specific server.

This service only maintains one entity-tag, hence one entity-tag per context. If the application wants to issue another publication using a different entity-tag, it must create another context.

This service reports events to the application through the ISipPublishMgr (see page 513) interface.

The ISipPublishSvc is an ECOM object

The ISipPublishSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipPublishSvc

Interface Id: IID_ISipPublishSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipPublishSvc.h

See Also

ISipPublishMgr (see page 513)

Methods

Method	Description
◆ A GetEntityTag (see page 517)	Retrieves the entity-tag corresponding the the current publication.
◆ A Modify (see page 518)	Modifies the publication.
◆ A Publish (see page 518)	Sends an initial publication to an ESC.
◆ A Refresh (see page 519)	Refreshes the publication expiration value
◆ A Remove (see page 520)	Removes the publication by sending a PUBLISH request with an Expires header with a value of zero.
◆ A SetDefaultExpiration (see page 520)	Sets the default expiration value for a publication.
◆ A SetEntityTag (see page 521)	Sets the entity-tag to use in the next PUBLISH requests.
◆ A SetExpiringThreshold (see page 521)	Sets the delay before expiration at which the EvExpiring event fires.
◆ A SetManager (see page 522)	Configures the manager of this service.

Legend

◆	Method
A	abstract

3.7.1.15.1 - Methods

3.7.1.15.1.1 - ISipPublishSvc::GetEntityTag Method

Retrieves the entity-tag corresponding the the current publication.

C++

```
virtual mxt_result GetEntityTag(OUT CString& rstrEntityTag) = 0;
```

Parameters

Parameters	Description
OUT CString& rstrEntityTag	The entity-tag found in the SIP-ETag header.

Returns

resS_OK if a 2xx response has been received and it contained an entity-tag or if an entity-tag was set with the SetEntityTag (see page 521) method.

resFE_INVALID_STATE if there is no manager associated with this service. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

Description

Retrieves the entity-tag set via the SetEntityTag (see page 521) method or received in a 2xx response after a PUBLISH request has been issued by this service.

See Also

ISipPublishMgr::EvSuccess (see page 516), SetEntityTag (see page 521)

3.7.1.15.1.2 - ISipPublishSvc::Modify Method

Modifies the publication.

C++

```
virtual mxt_result Modify(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	An opaque parameter to the transaction.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message body that is put in the PUBLISH request to be sent. According to the specification, an initial PUBLISH MUST contain a message body. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager associated with this service, if a final response to a previous request has not been received, or if an initial PUBLISH request has not been sent. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK if the request has been correctly sent.

resFE_FAIL otherwise.

Description

This method implements the PUBLISH request modify operation for a publication from an EPA to an ESC. Its purpose is to change the state of an existing publication.

The entity-tag received in a 2xx response for a previous PUBLISH request or set via the SetEntityTag (see page 521) method is placed in the SIP-If-Match header of the PUBLISH request issued by this method. The event name and the expires value are also placed in the headers of this PUBLISH modify message.

The Modify method API allows a modify operation with a NULL message body. According to RFC 3903, it MUST contain a message body. This message body holds the state information for the publication.

See Also

ISipPublishMgr (see page 513), Publish (see page 518), Refresh (see page 519), Remove (see page 520)

3.7.1.15.1.3 - ISipPublishSvc::Publish Method

Sends an initial publication to an ESC.

C++

```
virtual mxt_result Publish(IN mxt_opaque opqTransaction, IN unsigned int uExpiresS, IN const CString& rstrEvent,
```

```
IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	An opaque parameter to the transaction.
IN unsigned int uExpiresS	The number of seconds that this publication is to remain on the server. Note that the server may override this in its response. This value is later used when refreshing or modifying a publication.
IN const CString& rstrEvent	The value of the Event header. This same value is later used for refreshes and modifications to the publication.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message body that is put in the PUBLISH request to be sent. According to the specification, an initial PUBLISH MUST contain a message body. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager associated with this service or if a final response to a previous request has not been received. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK if the request has been correctly sent.

resFE_FAIL otherwise.

Description

This method implements the PUBLISH request initial operation for a publication from an EPA to an ESC.

The Publish method API allows an initial PUBLISH with a NULL message body. According to RFC 3903, it MUST include a message body with a published event name and MAY have an Expires header field. The message body contains the state of the publication.

If the publish succeeds, the server issues a 2xx response with an entity-tag and an Expires header. If, for any reason, the Expires header is missing in the 2xx response, the service assumes a default value of 1 hour for its publication expiration.

See Also

ISipPublishMgr (see page 513), Modify (see page 518), Refresh (see page 519), Remove (see page 520)

3.7.1.15.1.4 - ISipPublishSvc::Refresh Method

Refreshes the publication expiration value

C++

```
virtual mxt_result Refresh(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	An opaque parameter to the transaction.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.
rstrEntityTag	The value of the SIP-If-Match header.

Returns

resFE_INVALID_STATE if there is no manager associated with this service, if a final response to a previous request has not been received, or if an initial PUBLISH request has not been sent. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK if the request has been correctly sent.

resFE_FAIL otherwise.

Description

This method implements the PUBLISH request refresh operation for a publication from an EPA to an ESC. Its purpose is to extend the expiration value for an existing publication.

The entity-tag received in a 2xx response for a previous PUBLISH request or set via the SetEntityTag (see page 521) method is placed in the SIP-If-Match header of the PUBLISH request issued by this method. The event name and the expires value are also placed in the headers of this PUBLISH refresh message.

See Also

ISipPublishMgr (see page 513), Publish (see page 518), Modify (see page 518), Remove (see page 520)

3.7.1.15.1.5 - ISipPublishSvc::Remove Method

Removes the publication by sending a PUBLISH request with an Expires header with a value of zero.

C++

```
virtual mxt_result Remove(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	An opaque parameter to the transaction.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager associated with this service, if a final response to a previous request has not been received, or if an initial PUBLISH request has not been sent. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method implements the PUBLISH request remove operation for a publication from an EPA to an ESC. Removing a publication from an Esc is done as a refresh with an expires value set to 0. The last entity-tag received in 2xx or set via SetEntityTag (see page 521) is also placed in the SIP-If-Match header to match the publication at the ESC level.

See Also

ISipPublishMgr (see page 513), Publish (see page 518), Modify (see page 518), Refresh (see page 519)

3.7.1.15.1.6 - ISipPublishSvc::SetDefaultExpiration Method

Sets the default expiration value for a publication.

C++

```
virtual mxt_result SetDefaultExpiration(IN unsigned int uDefaultExpirationInS) = 0;
```

Parameters

Parameters	Description
IN unsigned int uDefaultExpirationInS	The default expiration value.

Returns

resFE_INVALID_STATE if there is no manager associated with this service. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK otherwise.

Description

This function configures the default lifetime for a published event. The default expiration value is by default set to 1 hour. This default value is only used by the Publish (see page 518) service if the Expires header is invalid or missing in a 2xx response to a PUBLISH request.

See Also

ISipPublishMgr::EvExpiring (see page 514), ISipPublishMgr::EvExpired (see page 514)

3.7.1.15.1.7 - ISipPublishSvc::SetEntityTag Method

Sets the entity-tag to use in the next PUBLISH requests.

C++

```
virtual mxt_result SetEntityTag(IN const CString& rstrEntityTag) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEntityTag	The entity-tag to put in SIP-If-Match headers.

Returns

resFE_INVALID_STATE if there is no manager associated with this service. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resFE_INVALID_ARGUMENT if the rstrEntityTag param is empty

resS_OK otherwise.

Description

Sets the entity-tag to be used in the next PUBLISH requests. This is true for the Publish (see page 518), Modify (see page 518), Refresh (see page 519), and Remove (see page 520) methods.

See Also

GetEntityTag (see page 517), Publish (see page 518), Modify (see page 518), Refresh (see page 519), Remove (see page 520)

3.7.1.15.1.8 - ISipPublishSvc::SetExpiringThreshold Method

Sets the delay before expiration at which the EvExpiring event fires.

C++

```
virtual mxt_result SetExpiringThreshold(IN unsigned int uDelayInS) = 0;
```

Parameters

Parameters	Description
IN unsigned int uDelayInS	The delay between the EvExpiring and the expiration in seconds.

Returns

resFE_INVALID_STATE if there is no manager associated with this service. No manager is associated with the service if none was set with SetManager (see page 522) or if Clear was called on the IContext interface.

resS_OK otherwise.

Description

Configures the delay between the moment ISipPublishMgr::EvExpiring (see page 514) event is generated and the publication expires. By default, this delay is 60 seconds. Note that the event EvExpiring is generated as soon as the time before expiration becomes lower than that threshold. The exception to that rule is that even if the ESC returns an expiration time that is lower than the threshold, this service does NOT generate an EvExpiring event. Instead, the manager of the publish service is warned only by an EvExpired event when the published information has expired. Also, if the threshold is set to 0, only the EvExpired event is generated.

This method affects only future publications. This means that if there is an active publication while this method is called, the EvExpiring

event is called according to the previous threshold value.

If the expires value is smaller than or equal to the threshold, the EvExpiring event is never called, EvExpired is instead called directly.

See Also

ISipPublishMgr::EvExpiring (see page 514)

3.7.1.15.1.9 - ISipPublishSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipPublishMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipPublishMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.16 - ISipRedirectionMgr Class

Class Hierarchy

```
[ISipRedirectionMgr]
```

C++

```
class ISipRedirectionMgr;
```

Description

The Redirection Event Manager is the interface through which the Redirection Service talks to the application. The stack informs the application through this interface that the user it is trying to reach can be found at a different location.

Location

SipUserAgent/ISipRedirectionMgr.h

See Also

ISipRedirectionSvc (see page 523)

Destructors

Destructor	Description
~ISipRedirectionMgr (see page 523)	<< Destructors >>

Legend

	Method
	virtual

Methods

Method	Description
EvRedirected (see page 523)	Notifies the application that the party it is trying to reach can be reached at another URL.

Legend

	Method
	abstract

3.7.1.16.1 - Constructors**3.7.1.16.1.1 - ISipRedirectionMgr::~ISipRedirectionMgr Destructor**

<< Constructors >>

C++

```
virtual ~ISipRedirectionMgr();
```

3.7.1.16.2 - Methods**3.7.1.16.2.1 - ISipRedirectionMgr::EvRedirected Method**

Notifies the application that the party it is trying to reach can be reached at another URL.

C++

```
virtual void EvRedirected(IN ISipRedirectionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipRedirectionSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality. The request can be reissued (with RelissueRequest) through this interface.
IN const CSipPacket& rPacket	The packet that triggered a redirection.

Description

Notifies the application that the party it is trying to reach is unreachable at the last URL tried but other possible URLs remain. The first time such an event occurs in a context is upon receiving a 3xx response to an outgoing request. It then occurs every time a new target has failed and that there are more targets to try.

When this event occurs, the application should use the redirection service UseNextContact or the UseContact method and re-issue the request with pClientEventCtrl.

This event never occurs with a 6xx response packet since such a response code means that the last tried target has authoritative information that the request will fail on every possible target. In this case, all the untried URLs remembered by the redirection service are cleared. This means that as soon as a 6xx response is received, there will not be any event generated (not even when the 6xx response is received), until another 3xx response is received.

See Also

ISipRedirectionSvc (see page 523)

3.7.1.17 - ISipRedirectionSvc Class**Class Hierarchy****C++**

```
class ISipRedirectionSvc : public IEComUnknown;
```

Description

The Redirection Service is used to help the application manage recursive searches for a remote user. When the application sends a request and receives a 3xx class response, this service manages the various contacts that are received in such a response and it lets

the application know through an event that there are other locations where the remote user can be reached.

This service manages the contact list in such a way as to avoid falling in an endless redirection loop. Thus, when searching for a user and receiving multiple 3xx responses, this service only keeps the contacts that were not tried in its contact list. All contacts where the application has sent a request are removed from the list.

To reach a contact that is not in the contacts list of the redirection service, the application can use the SetCurrentTarget method on the user agent service and then re-issue the request. The request is then sent to the contact location as set in the current target.

The Redirection Service can handle 3xx responses for any type of request.

This service reports events to the application through the ISipRedirectionMgr (see page 522) interface.

The ISipRedirectionSvc is an ECOM object

The ISipRedirectionSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipRedirectionSvc

Interface Id: IID_ISipRedirectionSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Updated Behaviour:

The behaviour of the redirection service has changed from MxSF v3.x.

- In v4.0, the contacts are removed from the list when a request is sent using this contact as Request-URI.
- In v4.0, when a contact is received twice in a 3xx response, the contact version with the maximum qvalue is kept. When there is no qvalue in a contact, 1.0 is taken as qvalue.
- The responses received from requests made inside of a dialog are not monitored by the service.
- Even if a 2xx response is received, the contacts that were not tried are no longer cleared so this service sends an EvRedirected as soon as it receives a negative response to a request made outside of a dialog, if there are still contacts to try.

Location

SipUserAgent/ISipRedirectionSvc.h

See Also

ISipRedirectionMgr.h

Methods

Method	Description
◆ A GetContacts (see page 525)	Retrieves the list of possible contacts where the parent context's target could be reached.
◆ A RemoveContact (see page 525)	Removes the specified contact from the list.
◆ A SetManager (see page 525)	Configures the manager of this service.
◆ A UseContact (see page 526)	Makes the parent context use the URI at position uContactIndex in the order of preference for the request-URI of future requests.
◆ A UseNextContact (see page 526)	Makes the parent context use the Contact header with the highest 'q value' for the request-URI of future requests.

Legend

◆	Method
A	abstract

3.7.1.17.1 - Methods

3.7.1.17.1.1 - ISipRedirectionSvc::GetContacts Method

Retrieves the list of possible contacts where the parent context's target could be reached.

C++

```
virtual const CSipHeader* GetContacts() = 0;
```

Returns

A header that contains the contacts ordered by decreasing preference. If there are more than one contact, they are chained in the header and they can be accessed via `CSipHeader::GetNextHeader` (see page 265).

NULL if there is no URI to try.

Note that a copy of the header chain must be done if the application wants to keep a modified copy of the chain.

Description

This method returns the list of contacts where the target of the parent context could be reached. It is ordered by decreasing 'q' value so the first contact in the list is the preferred one.

If there is no such contact, NULL is returned. In this case, calling `UseNextContact` (see page 526) or `UseContact` (see page 526) yields an error.

See Also

`UseContact` (see page 526), `UseNextContact` (see page 526)

3.7.1.17.1.2 - ISipRedirectionSvc::RemoveContact Method

Removes the specified contact from the list.

C++

```
virtual mxt_result RemoveContact(IN unsigned int uContactIndex) = 0;
```

Parameters

Parameters	Description
IN unsigned int uContactIndex	The specified index in the contact list. 0-based where 0 refers to the header containing all the others.

Returns

`resFE_INVALID_STATE` if the manager was not set OR if `Clear` was called on the context OR if there is no contact to try.

`resFE_INVALID_ARGUMENT` if `uContactIndex` is invalid.

`resS_OK` otherwise.

Description

This method removes the contact from the list of contacts to try given by `GetContacts` (see page 525). The contacts that have been removed are no longer suggested.

If the specified index does not exist or if there is no contact to try, the method returns an error.

See Also

`GetContacts` (see page 525)

3.7.1.17.1.3 - ISipRedirectionSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipRedirectionMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipRedirectionMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that a manager MUST be associated with this service before it is used. If no manager is associated with the service, the service will not monitor any request nor response.

3.7.1.17.1.4 - ISipRedirectionSvc::UseContact Method

Makes the parent context use the URI at position uContactIndex in the order of preference for the request-URI of future requests.

C++

```
virtual mxt_result UseContact(IN unsigned int uContactIndex) = 0;
```

Parameters

Parameters	Description
IN unsigned int uContactIndex	The index of the contact to use in the vector of contacts. 0-based where 0 refers to the header containing all the others.

Returns

resFE_INVALID_STATE if the manager was not set OR if Clear was called on the context OR if there is no other contact to try OR if there is no ISipUserAgentSvc (see page 639) attached to the context to which this service is attached.

resFE_INVALID_ARGUMENT if uContactIndex is not a valid index.

resFE_FAIL if changing the request-URI failed.

resS_OK if the request-URI was successfully changed.

Description

This method updates the state of the parent context so it uses the chosen URI as the request-URI of future requests.

It returns an error if there is no such contact to use, that is the index is greater than the number given by CSipHeader::GetNbNextHeaders (see page 264) on the CSipHeader (see page 239) given by GetContacts (see page 525) or if there is no contact to try.

The application should call this method only if it has positive knowledge that the preferred contact is not the target to which it wants to send a request. Called with a parameter of 0, this method is totally equivalent to UseNextContact (see page 526).

Usually, after calling UseContact, the application calls RelssueRequest on the ISipClientEventControl (see page 19) pointer given by the EvRedirected event to re-send the request to the new request-URI.

See Also

UseNextContact (see page 526), GetContacts (see page 525)

3.7.1.17.1.5 - ISipRedirectionSvc::UseNextContact Method

Makes the parent context use the Contact header with the highest 'q value' for the request-URI of future requests.

C++

```
virtual mxt_result UseNextContact() = 0;
```

Returns

resFE_INVALID_STATE if the manager was not set OR if Clear was called on the context OR if there is no other contact to try OR if there is no ISipUserAgentSvc (see page 639) attached to the context to which this service is attached.

resFE_FAIL if changing the request-URI failed.

resS_OK if the request-URI was successfully changed.

Description

This method updates the state of the parent context so it uses the URL of the contact that has the highest 'q value' as the request-URI of future requests. It returns an error if there are no contacts to use. This error condition usually happens if there were no redirections or if all possible contacts were already tried.

Usually, after calling UseNextContact, the application calls RelissueRequest on the ISipClientEventControl (see page 19) pointer given by the EvRedirected event to re-send the request to the new request-URI.

See Also

UseContact (see page 526)

3.7.1.18 - ISipRefereeMgr Class**Class Hierarchy**

 ISipRefereeMgr

C++

```
class ISipRefereeMgr;
```

Description

The interface through which the ISipRefereeSvc (see page 531) reports events to the application.

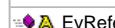
Location

SipUserAgent/ISipRefereeMgr.h

See Also

ISipRefereeSvc (see page 531)

Methods

Method	Description
 EvExpired (see page 528)	The subscription is expired.
 EvFailure (see page 528)	A failure (<= 3xx) has been received for a NOTIFY request sent by the ISipRefereeSvc (see page 531).
 EvInvalidRequest (see page 528)	A request has been automatically rejected by this service.
 EvProgress (see page 529)	A provisional response (1xx) has been received for a NOTIFY request sent by the ISipRefereeSvc (see page 531).
 EvReferred (see page 529)	A REFER request has been received.
 EvRefreshed (see page 530)	A SUBSCRIBE request to refresh an active subscription has been received.
 EvSuccess (see page 530)	A success response (2xx) has been received for a NOTIFY request sent by the ISipRefereeSvc (see page 531).
 EvTerminated (see page 531)	A SUBSCRIBE request to terminate an active/pending subscription has been received.

Legend

	Method
	abstract

3.7.1.18.1 - Methods

3.7.1.18.1.1 - ISipRefereeMgr::EvExpired Method

The subscription is expired.

C++

```
virtual void EvExpired(IN ISipRefereeSvc* pSvc, IN mxt_opaque opqReferId) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface that has an expired subscription.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.

Description

Notifies the application that a subscription is expired. This only happens for subscriptions created by accepting a REFER request through EvReferred (see page 529) and not call SendFinalReferralStatus for that subscription.

When this event occurs, the application should call SendFinalReferralStatus.

3.7.1.18.1.2 - ISipRefereeMgr::EvFailure Method

A failure (<= 3xx) has been received for a NOTIFY request sent by the ISipRefereeSvc (see page 531).

C++

```
virtual void EvFailure(IN ISipRefereeSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the NOTIFY request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.
IN const CSipPacket& rResponse	The failure response received.

Description

Notifies the application that a failure response (<=300) has been received for a NOTIFY request that it sent. Note that this NOTIFY request could result from a call to SendReferralStatus or SendFinalReferralStatus.

When the NOTIFY request was sent through the SendReferralStatus method and the application does not know how to fix the error condition, it should call SendFinalReferralStatus.

See Also

ISipClientEventControl (see page 19), ISipRefereeSvc (see page 531)

3.7.1.18.1.3 - ISipRefereeMgr::EvInvalidRequest Method

A request has been automatically rejected by this service.

C++

```
virtual void EvInvalidRequest(IN ISipRefereeSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest, IN mxt_result reason) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the request has been received.

IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rRequest	The request received. It can be either a SUBSCRIBE or a REFER.
IN mxt_result reason	<ul style="list-style-type: none"> resFE_MISSING_HEADER: The REFER request is missing a valid Refer-To header. A "400 Bad Request" response has already been sent. resFE_UNKNOWN_EVENT: The SUBSCRIBE request is missing an Event header or the event type is not "refer". A "489 Bad Event" response has already been sent. resFE_UNKNOWN_SUBSCRIPTION: The SUBSCRIBE request did not correspond to any REFER sent by this service. A "481 Subscription does not exist" response has already been sent. resFE_EXPIRED_SUBSCRIPTION: The SUBSCRIBE has been received after a NOTIFY with "subscription-state" "terminated" was sent or after the expiration time. A "481 Subscription does not exist" response has already been sent.

Description

Notifies the application that an invalid SUBSCRIBE or REFER has been received. Usually, the application has no action to take other than reporting this invalid packet if wanted. The ISipRefereeSvc (see page 531) that reported the event already took care of sending the appropriate response.

One action that can be taken is to destroy the context if it was created only to handle this SUBSCRIBE or REFER as it will not create a dialog.

3.7.1.18.1.4 - ISipRefereeMgr::EvProgress Method

A provisional response (1xx) has been received for a NOTIFY request sent by the ISipRefereeSvc (see page 531).

C++

```
virtual void EvProgress(IN ISipRefereeSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the NOTIFY request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.
IN const CSipPacket& rResponse	The provisional response received.

Description

Notifies the application that a provisional response has been received for a NOTIFY request that it sent. Note that this NOTIFY request could result from a call to SendReferralStatus or SendFinalReferralStatus.

See Also

ISipClientEventControl (see page 19), ISipRefereeSvc (see page 531)

3.7.1.18.1.5 - ISipRefereeMgr::EvReferred Method

A REFER request has been received.

C++

```
virtual void EvReferred(IN ISipRefereeSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN mxt_opaque opqReferId, IN const CNameAddr& rReferToAddr, IN const CSipPacket& rRefer) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the REFER request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN mxt_opaque opqReferId	The opaque id that the application must use to send NOTIFY for that REFER. It is also put in the events issued by this service that relate to this REFER.
IN const CNameAddr& rReferToAddr	The address found in the Refer-To header of the received REFER request.
IN const CSipPacket& rRefer	The received REFER request.

Description

Receives a REFER request. This request is guaranteed to contain a valid Refer-To header. When receiving this event, the manager can send any final response through the pServerEventControl interface.

If a success response (2xx) is sent, the manager should immediately call SendReferralStatus to send a NOTIFY to establish the subscription. When the manager does not intend to continue the subscription, it can call SendFinalReferralStatus.

Care should be taken when calling SendReferralStatus to use a long enough expiration value because it will not be able to increase this value later. Only the sender of the REFER can extend the subscription by sending a SUBSCRIBE request.

See Also

ISipRefereeSvc ([see page 531](#))

3.7.1.18.1.6 - ISipRefereeMgr::EvRefreshed Method

A SUBSCRIBE request to refresh an active subscription has been received.

C++

```
virtual void EvRefreshed(IN ISipRefereeSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN mxt_opaque opqReferId, IN unsigned int uExpirationSec, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.
IN unsigned int uExpirationSec	The expiration time suggested in the SUBSCRIBE request. If the request does not suggest an expiration time, the default expiration time of 60 seconds is used.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

Description

This method is called when a valid SUBSCRIBE request is received for a subscription that already exists. Accepting this request with a success response extends the subscription on which the application can call SendReferralStatus or SendFinalReferralStatus.

The application must respond quickly to that request.

See Also

ISipRefereeSvc ([see page 531](#))

3.7.1.18.1.7 - ISipRefereeMgr::EvSuccess Method

A success response (2xx) has been received for a NOTIFY request sent by the ISipRefereeSvc ([see page 531](#)).

C++

```
virtual void EvSuccess(IN ISipRefereeSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the NOTIFY request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest is usually called to retry failed requests instead of successful ones.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.
IN const CSipPacket& rResponse	The success response received.

Description

Notifies the application that a success response has been received for a NOTIFY request that it sent. If that NOTIFY request was sent through SendReferralStatus, the application should consider that the subscription is still active. If the NOTIFY request was sent through SendFinalReferralStatus, the application should consider that this subscription is terminated. It should not call SendFinalReferralStatus nor SendReferralStatus again on that subscription.

See Also

ISipClientEventControl (see page 19), ISipRefereeSvc (see page 531)

3.7.1.18.1.8 - ISipRefereeMgr::EvTerminated Method

A SUBSCRIBE request to terminate an active/pending subscription has been received.

C++

```
virtual void EvTerminated(IN ISipRefereeSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN mxt_opaque opqReferId, IN const CSipPacket& rSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc* pSvc	The interface on which the SUBSCRIBE request has been received.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN mxt_opaque opqReferId	The opaque id that accompanied the corresponding REFER request when EvReferred (see page 529) was called.
IN const CSipPacket& rSubscribe	The SUBSCRIBE request received.

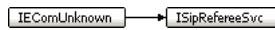
Description

This method is called when a SUBSCRIBE request with "Expires" header set to 0 is received for an active or pending subscription.

If the application accepts this request, it should then call SendFinalReferralStatus to send a final NOTIFY.

See Also

ISipRefereeSvc (see page 531)

3.7.1.19 - ISipRefereeSvc Class**Class Hierarchy****C++**

```
class ISipRefereeSvc : public IEComUnknown;
```

Description

This interface is used to act as a referee as defined in RFC 3515. It lets the application receive a REFER request and send the proper

notification of the request progress.

It reports events received to the application through the `ISipRefereeMgr` (see page 527) interface.

The `ISipRefereeSvc` is an ECOM object

The `ISipRefereeSvc` is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipRefereeSvc

Interface Id: IID_ISipRefereeSvc

A user can query the `ISipContext` (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the `ISipContext` (see page 23) through the same mean.

Location

`SipUserAgent/ISipRefereeSvc.h`

See Also

`ISipRefereeMgr` (see page 527)

Methods

Method	Description
ConfigureNotifyIdParameterUsage (see page 532)	Configures whether the stack uses the "id" parameter in the Event header of NOTIFY requests for the first REFER request.
SendFinalReferralStatus (see page 533)	Sends a final NOTIFY for a received REFER.
SendReferralStatus (see page 534)	Sends a NOTIFY for a received REFER.
SetManager (see page 535)	Configures the manager of this service.

Legend

	Method
	abstract

Enumerations

Enumeration	Description
<code>EIdParameterUsage</code> (see page 535)	

3.7.1.19.1 - Methods

3.7.1.19.1.1 - `ISipRefereeSvc::ConfigureNotifyIdParameterUsage` Method

Configures whether the stack uses the "id" parameter in the Event header of NOTIFY requests for the first REFER request.

C++

```
virtual void ConfigureNotifyIdParameterUsage( IN EIdParameterUsage eIdParamUsage ) = 0;
```

Parameters

Parameters	Description
<code>IN EIdParameterUsage eIdParamUsage</code>	<ul style="list-style-type: none"> <code>eID_PARAM_ALWAYS_PRESENT</code>: The id parameter is added to all NOTIFYs. This is the default behaviour. <code>eID_PARAM_ABSENT_FOR_FIRST_REFERER</code>: The id parameter in the NOTIFYs to the first REFER are not set.

Description

Configures whether the stack uses the "id" parameter in the Event header of the NOTIFY it sends after a subscription is created by receiving a REFER. This is used only for the NOTIFY(s) associated with the first REFER received by this service.

RFC 3515 states that the NOTIFY(s) associated with the first REFER sent on a dialog MAY contain an "id" parameter in the Event header, and the NOTIFY(s) associated with any additional REFER after the first one MUST contain this parameter. The default behavior of the stack is to always include the parameter for all NOTIFY(s) it sends, however this can sometimes cause interoperability issues. An application can use this method to configure whether the stack includes the "id" parameter for the NOTIFY(s) associated with the first REFER it has received. All NOTIFYs associated with subsequent REFERs will always contain the "id" parameter in the Event header.

See Also

ISipRefereeSvc (see page 531)

3.7.1.19.1.2 - ISipRefereeSvc::SendFinalReferralStatus Method

Sends a final NOTIFY for a received REFER.

C++

```
virtual mxt_result SendFinalReferralStatus(IN mxt_opaque opqReferId, IN ISipNotifierSvc::EReason eReason, IN
unsigned int uRetryAfterSec, IN const CSipStatusLine& rContentStatusLine, IN const CHeaderList* pContentHeaders,
IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqReferId	The opaque id that accompanied the received REFER.
IN ISipNotifierSvc::EReason eReason	The "reason" parameter to put in the "Subscription-State" header. When the value is eNO_REASON, the parameter is not put in the "Subscription-State" header.
IN unsigned int uRetryAfterSec	The number of seconds to put in the "retry-after" parameter of the "Subscription-State" header. When the value is 0, no "retry-after" parameter is put. Note that RFC 3265 defines semantics for this parameter only if the eReason parameter has the value eNO_REASON, ePROBATION, or eGIVEUP.
IN const CSipStatusLine& rContentStatusLine	The status-line to put in the content of the NOTIFY request. It is the actual referral status. Note that this response can be either final or provisional. The fact that this NOTIFY is final simply indicates to the referrer that the referee no longer informs it of refer request progress.
IN const CHeaderList* pContentHeaders	Headers to put in the content after the status-line. RFC 3515 does not define semantics for these headers. However, care should be taken when populating this parameter. Refer to the security section of RFC 3515 for details. It can be NULL, which means that only the request-line is sent in the content.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN. This parameter puts the extra headers in the NOTIFY request. To put headers in the content of the NOTIFY, use the pContentHeaders parameter.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

- **resFE_INVALID_ARGUMENT:** The id does not correspond to any received REFER in this object.
- **resFE_INVALID_STATE:** The attached ISipUserAgentSvc (see page 639) is incorrectly configured. For instance, the contact list could be empty.
- **resFE_FAIL:** The NOTIFY request could not be sent.
- **resS_OK:** The NOTIFY request has been successfully sent.

Description

Sends a NOTIFY with the "Subscription-State" header set to "terminated". The "message/sipfrag" content mandated by RFC 3515 is created from the rContentStatusLine and the pContentHeaders parameters.

This method should be called when the event `EvExpired` is called by the `ISipRefereeMgr` (see page 527).

It should also be called when accepting a `SUBSCRIBE` received through the `EvTerminated` event of the `ISipRefereeMgr` (see page 527).

Finally, it should be called when the application knows that it will no longer report status for this referral.

See Also

`SendReferralStatus` (see page 534)

3.7.1.19.1.3 - `ISipRefereeSvc::SendReferralStatus` Method

Sends a `NOTIFY` for a received `REFER`.

C++

```
virtual mxt_result SendReferralStatus(IN mxt_opaque opqReferId, IN ISipNotifierSvc::EState eSubscriptionState,
IN unsigned int uExpirationSec, IN const CSipStatusLine& rContentStatusLine, IN const CHeaderList*
pContentHeaders, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*&
rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqReferId	The opaque id that accompanied the received <code>REFER</code> .
IN ISipNotifierSvc::EState eSubscriptionState	The state of the subscription to put in the <code>Subscription-State</code> header of the <code>NOTIFY</code> to send. It can be either <code>ePENDING</code> or <code>eACTIVE</code> .
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the current remaining expiration for that subscription is used. This value is put in the "Subscription-State" header's "expires" parameter of the <code>NOTIFY</code> request sent. The application should not use 0 the first time it calls this method since the initial expiration is set really high so the application can decrease it to what it wants. This expiration should be set to the maximum time expected to obtain a final status for the referred request.
IN const CSipStatusLine& rContentStatusLine	The status-line to put in the content of the <code>NOTIFY</code> request. It is the actual referral status. Note that it can be a final response when other responses are still expected (because the application will retry the referred request for instance). When no other responses are expected, <code>SendFinalReferralStatus</code> (see page 533) should be used instead.
IN const CHeaderList* pContentHeaders	Headers to put in the content after the status-line. RFC 3515 does not define semantics for these headers. However, care should be taken when populating this parameter. Refer to the security section of RFC 3515 for details. It can be <code>NULL</code> , which means that only the request-line is sent in the content.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be <code>NULL</code> , which means that no extra headers are added. Ownership of this parameter is <code>TAKEN</code> . This parameter puts the extra headers in the <code>NOTIFY</code> request. To put headers in the content of the <code>NOTIFY</code> , use the <code>pContentHeaders</code> parameter.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

- `resFE_INVALID_ARGUMENT`: The id does not correspond to any `REFER` received in this object or the expiration time is higher than the remaining time for that subscription.
- `resFE_INVALID_STATE`: The id corresponds to a subscription that is expired. Use `SendFinalReferralStatus` (see page 533) instead to send a final `NOTIFY`. Note that this is the case when receiving a `SUBSCRIBE` with a "Expires" header value of 0 OR the attached `ISipUserAgentSvc` (see page 639) is incorrectly configured. For instance, the contact list could be empty.
- `resFE_FAIL`: The `NOTIFY` request could not be sent.
- `resS_OK`: The `NOTIFY` request has been successfully sent.

Description

Sends a NOTIFY request for an active or pending subscription. The "message/sipfrag" content mandated by RFC 3515 is created from the rContentStatusLine and the pContentHeaders parameters.

The application should use a non 0 uExpirationSec the first time it calls this method in order to set the initial expiration time for the subscription. Otherwise, the default expiration time is used. Note that once an expiration time is set, the referee cannot increase it. The subscription can only be extended by the referrer.

When the application lowers the expiration time by using a non NULL value in the uExpirationSec parameter, the expiration time is lowered if the NOTIFY request receives a success response.

See Also

[SendFinalReferralStatus](#) (see page 533)

3.7.1.19.1.4 - ISipRefereeSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipRefereeMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.19.2 - Enumerations

3.7.1.19.2.1 - ISipRefereeSvc::EIdParameterUsage Enumeration

```
enum EIdParameterUsage {
    eID_PARAM_ALWAYS_PRESENT,
    eID_PARAM_ABSENT_FOR_FIRST_REFER
};
```

Description

Determines if the referee service should add the 'id' parameter to the Event header.

Members

Members	Description
eID_PARAM_ALWAYS_PRESENT	Parameter is always present.
eID_PARAM_ABSENT_FOR_FIRST_REFER	Parameter is absent for first REFER.

3.7.1.20 - ISipReferrerMgr Class

Class Hierarchy

```
ISipReferrerMgr
```

C++

```
class ISipReferrerMgr;
```

Description

The interface through which the ISipReferrerSvc (see page 542) reports events to the application.

Location

SipUserAgent/ISipReferrerMgr.h

See Also

ISipReferrerMgr

Methods

Method	Description
◆ A EvExpired (see page 536)	The subscription is expired.
◆ A EvExpiring (see page 537)	The subscription is about to expire.
◆ A EvIntervalTooSmall (see page 537)	An "Interval Too Small" (423) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).
◆ A EvInvalidNotify (see page 537)	An invalid NOTIFY has been received.
◆ A EvReferFailure (see page 538)	A failure (<= 3xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).
◆ A EvReferProgress (see page 539)	A provisional response (1xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).
◆ A EvReferStatus (see page 539)	Notifies the application that a NOTIFY for a REFER sent has been received.
◆ A EvReferSuccess (see page 540)	A success response (2xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).
◆ A EvSubscribeFailure (see page 540)	A failure (<= 3xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).
◆ A EvSubscribeProgress (see page 541)	A provisional response (1xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).
◆ A EvSubscribeSuccess (see page 541)	A success response (2xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).

Legend

◆	Method
◆ A	abstract

Enumerations

Enumeration	Description
EState (see page 542)	

3.7.1.20.1 - Methods

3.7.1.20.1.1 - ISipReferrerMgr::EvExpired Method

The subscription is expired.

C++

```
virtual void EvExpired(IN ISipReferrerSvc* pSvc, IN mxt_opaque opqReferId) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface that has an expired subscription.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.

Description

Informs the application that a subscription is expired. There is no action to take by the application as far as the ISipReferrerSvc (see page 542) is concerned.

See Also

EvExpiring (see page 537)

3.7.1.20.1.2 - ISipReferrerMgr::EvExpiring Method

The subscription is about to expire.

C++

```
virtual void EvExpiring(IN ISipReferrerSvc* pSvc, IN mxt_opaque opqReferId) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface that has a subscription about to expire.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.

Description

Notifies the application that a subscription will expire in the amount of time configured with the SetExpiringThreshold method. This event does not occur when a subscription is set for a time smaller or equal to the threshold.

The application should then either call Refresh to keep the subscription active or Terminate to terminate it.

See Also

EvExpired (see page 536)

3.7.1.20.1.3 - ISipReferrerMgr::EvIntervalTooSmall Method

An "Interval Too Small" (423) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvIntervalTooSmall(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
unsigned int uMinExpirationSec, IN mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest would not correct the problem because the SUBSCRIBE request would be retried with the same expiration time.
IN unsigned int uMinExpirationSec	The minimum number of seconds acceptable for the subscription duration according to the remote peer.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The success response received.

Description

Notifies the application that a 423 Interval Too Small response has been received for a SUBSCRIBE request that it sent. This SUBSCRIBE was sent through Refresh. The application should call Refresh again but with an expiration time greater than or equal to uMinExpirationSec.

See Also

ISipClientEventControl (see page 19), ISipReferrerSvc (see page 542)

3.7.1.20.1.4 - ISipReferrerMgr::EvInvalidNotify Method

An invalid NOTIFY has been received.

C++

```
virtual void EvInvalidNotify(IN ISipReferrerSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket&
```

```
rNotify, IN mxt_result reason) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface that handled the notify.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rNotify	The NOTIFY request received.
IN mxt_result reason	<ul style="list-style-type: none"> resFE_UNKNOWN_EVENT: The Event header was missing or for an event type other than "refer". A "489 Bad Event" response has already been sent. resFE_UNKNOWN_SUBSCRIPTION: The NOTIFY does not correspond to a REFER sent by this service. A "481 Subscription does not exist" response has already been sent. resFE_EXPIRED_SUBSCRIPTION: The NOTIFY has been received after another NOTIFY with "subscription-state" "terminated" or after the expiration time. A "481 Subscription does not exist" response has already been sent. resFE_MISSING_HEADER: The received NOTIFY did not contain the mandatory "Subscription-State" header. resFE_INVALID_CONTENT: The received NOTIFY did not contain a valid "message/sipfrag".

Description

Informs the application that an invalid NOTIFY has been received. Usually, the application has no action to take other than reporting this invalid packet if wanted. The ISipReferrerMgr (see page 535) that reported the event already took care of sending the appropriate response.

One action that can be taken is to destroy the context if it was created only to handle this NOTIFY as it will not create a dialog.

See Also

EvNotified, EvTerminated

3.7.1.20.1.5 - ISipReferrerMgr::EvReferFailure Method

A failure (<= 3xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvReferFailure(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the REFER request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelssueRequest on it.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The failure response received.

Description

Notifies the application that a failure response (<=300) has been received for a REFER request that it sent. Note that this SUBSCRIBE request could result from a call to Subscribe, Fetch, Refresh, or Terminate.

Note that if the REFER request resulted from a call to RelssueRequest on a ISipClientEventControl (see page 19) interface, the opqReferId parameter is still the same as the one returned by the original call to the Refer method.

See Also

ISipClientEventControl (see page 19), ISipReferrerSvc (see page 542)

3.7.1.20.1.6 - ISipReferrerMgr::EvReferProgress Method

A provisional response (1xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvReferProgress(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the REFER request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The provisional response received.

Description

Notifies the application that a provisional response has been received for a REFER request that it sent. Note that if the REFER request resulted from a call to RelssueRequest on a ISipClientEventControl (see page 19) interface, the opqReferId parameter is still the same as the one returned by the original call to the Refer method.

See Also

ISipClientEventControl (see page 19), ISipReferrerSvc (see page 542)

3.7.1.20.1.7 - ISipReferrerMgr::EvReferStatus Method

Notifies the application that a NOTIFY for a REFER sent has been received.

C++

```
virtual void EvReferStatus(IN ISipReferrerSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN
mxt_opaque opqReferId, IN EState eState, IN const CSipStatusLine& rStatus, IN const CHeaderList*
pContentHeaders, IN const CSipPacket& rNotify) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the REFER was sent.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN EState eState	The state of the subscription. When eTERMINATED and the application accepts the NOTIFY by sending a success response, the subscription is terminated. This parameter is calculated from the Subscription-State header of the NOTIFY request received.
IN const CSipStatusLine& rStatus	The status line found in the content of the NOTIFY request.
IN const CHeaderList* pContentHeaders	Headers that accompanied the status line in the content of the NOTIFY. It can be NULL, which means that there were no such headers. Note that the application does not have ownership of this pointer and the header list must be copied if the application intends to use it after the method returned.
IN const CSipPacket& rNotify	The NOTIFY request received.

Description

Notifies the application that a valid NOTIFY request has been received. The ISipReferrerSvc (see page 542) has already validated that the NOTIFY is for an active subscription and that it is valid. It has also verified that the content is valid according to RFC 3515. It means that it was of type "message/sipfrag" and contained at least a valid status-line.

Note that this NOTIFY request is guaranteed to contain a valid "subscription-state" header.

See Also

ISipServerEventControl (see page 75)

3.7.1.20.1.8 - ISipReferrerMgr::EvReferSuccess Method

A success response (2xx) has been received for a REFER request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvReferSuccess(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the REFER request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest is usually called to retry failed requests instead of successful ones.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The success response received.

Description

Notifies the application that a success response has been received for a REFER request that it sent. The application should consider that the subscription is active and be ready to receive EvReferStatus (see page 539) events. Note that EvReferStatus (see page 539) can occur before EvReferSuccess when the initial NOTIFY is received before the REFER success response.

Note that if the REFER request resulted from a call to RelIssueRequest on a ISipClientEventControl (see page 19) interface, the opqReferId parameter is still the same as the one returned by the original call to the Refer method.

See Also

ISipClientEventControl (see page 19), ISipReferrerSvc (see page 542)

3.7.1.20.1.9 - ISipReferrerMgr::EvSubscribeFailure Method

A failure (<= 3xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvSubscribeFailure(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The failure response received.

Description

Notifies the application that a failure response (<=300) has been received for a SUBSCRIBE request that it sent. Note that this SUBSCRIBE request could result from a call to Refresh or Terminate.

See Also

[ISipClientEventControl](#) (see page 19), [ISipReferrerSvc](#) (see page 542)

3.7.1.20.1.10 - ISipReferrerMgr::EvSubscribeProgress Method

A provisional response (1xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvSubscribeProgress(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The provisional response received.

Description

Notifies the application that a provisional response has been received for a SUBSCRIBE request that it sent. Note that this SUBSCRIBE request could result from a call to Refresh or Terminate.

See Also

[ISipClientEventControl](#) (see page 19), [ISipReferrerSvc](#) (see page 542)

3.7.1.20.1.11 - ISipReferrerMgr::EvSubscribeSuccess Method

A success response (2xx) has been received for a SUBSCRIBE request sent by the ISipReferrerSvc (see page 542).

C++

```
virtual void EvSubscribeSuccess(IN ISipReferrerSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
mxt_opaque opqReferId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest is usually called to retry failed requests instead of successful ones.
IN mxt_opaque opqReferId	The opqReferId returned by the service when Refer was called.
IN const CSipPacket& rResponse	The success response received.

Description

Notifies the application that a success response has been received for a SUBSCRIBE request that it sent. If that SUBSCRIBE request was sent through Refresh, the application should consider that the subscription is active and be ready to receive EvReferStatus (see page 539) events. If the SUBSCRIBE request was sent through Terminate, the application should expect to receive the event EvTerminated or EvExpired (see page 536). Note that the EvTerminated or EvExpired (see page 536) can still occur unrequested if the referee decides to terminate the subscription or if the subscription expires.

See Also

[ISipClientEventControl](#) (see page 19), [ISipReferrerSvc](#) (see page 542)

3.7.1.20.2 - Enumerations

3.7.1.20.2.1 - ISipReferrerMgr::EState Enumeration

```
enum EState {
    ePENDING,
    eACTIVE,
    eTERMINATED
};
eACTIVE
eTERMINATED
```

Description

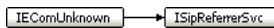
Subscription states.

Members

Members	Description
ePENDING	A subscription state.

3.7.1.21 - ISipReferrerSvc Class

Class Hierarchy



C++

```
class ISipReferrerSvc : public IEComUnknown;
```

Description

This interface is used to act as a referrer as defined in RFC 3515. It lets the application send a REFER request and receive the proper notification of the request progress.

It reports events received to the application through the ISipReferrerMgr (see page 535) interface.

The ISipReferrerSvc is an ECOM object

The ISipReferrerSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipReferrerSvc

Interface Id: IID_ISipReferrerSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipReferrerSvc.h

See Also

ISipReferrerMgr (see page 535)

Methods

Method	Description
Refer (see page 543)	Sends a REFER request.
Refresh (see page 543)	Extends the subscription for that REFER.
SetExpiringThreshold (see page 544)	Sets the time to warn the application before a subscription expires.
SetManager (see page 545)	Configures the manager of this service.
Terminate (see page 545)	Terminates the subscription for that REFER.

Legend

◆	Method
▲	abstract

3.7.1.21.1 - Methods

3.7.1.21.1.1 - ISipReferrerSvc::Refer Method

Sends a REFER request.

C++

```
virtual mxt_result Refer(IN const CNameAddr& rReferToAddr, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT mxt_opaque& ropqReferId, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rReferToAddr	The address to put in the Refer-To header.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REFER request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership of this parameter is TAKEN. RFC 3515 does not put any semantics in this content.
OUT mxt_opaque& ropqReferId	The id for that REFER request. It is useful when there are more than one REFER in the same context. This id is found in every event related to that REFER reported by the ISipReferrerMgr (see page 535). Note that this is an OUT parameter determined by the service. This id is also used as a parameter to the Refresh (see page 543) or the Terminate (see page 545) methods when needing to refresh or terminate the subscription associated with this refer. Note that this id is different from the id parameter of the Event header found in NOTIFY requests received and SUBSCRIBE requests sent for this REFER.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

- resFE_FAIL: The REFER request could not be sent.
- resFE_INVALID_STATE: The attached ISipUserAgentSvc (see page 639) is incorrectly configured. For instance, the contact list could be empty.
- resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
- resS_OK: The REFER request has been successfully sent.

Description

Sends a REFER request. This REFER request contains a Refer-To header with the rReferToAddr. The progress of the REFER request is reported through the ISipReferrerMgr (see page 535). At least EvReferSuccess or EvReferFailed is reported. When the REFER succeeds, the progression of the referral is reported through EvReferStatus. The subscription is terminated either when EvReferStatus is reported with eState set to eTERMINATED or when EvExpired is reported. Before a subscription terminates, the application can either decide to terminate it by calling Terminate (see page 545) or extend it by calling Refresh (see page 543).

See Also

ISipReferrerMgr (see page 535), Terminate (see page 545), Refresh (see page 543)

3.7.1.21.1.2 - ISipReferrerSvc::Refresh Method

Extends the subscription for that REFER.

C++

```
virtual mxt_result Refresh(IN mxt_opaque opqReferId, IN unsigned int uExpirationSec, IN mxt_opaque
opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT
ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqReferId	The opaque id corresponding to the REFER subscription to extend.
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration of 60 seconds is used.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership of this parameter is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

- **resFE_INVALID_ARGUMENT**: The id does not correspond to any subscription in this object.
- **resFE_FAIL**: The SUBSCRIBE request could not be sent.
- **resS_OK**: The SUBSCRIBE request has been successfully sent.

Description

This method sends a SUBSCRIBE request to extend the subscription created by the REFER it sent. It can be used when the subscription is about to expire and the final notification has not been received.

Note that when the application is no longer interested in the result of the referral process, it should call Terminate (see page 545).

See Also

Terminate (see page 545)

3.7.1.21.1.3 - ISipReferrerSvc::SetExpiringThreshold Method Updated behavior in 4.1.4

Sets the time to warn the application before a subscription expires.

C++

```
virtual void SetExpiringThreshold(IN unsigned int uThresholdSec) = 0;
```

Parameters

Parameters	Description
IN unsigned int uThresholdSec	The time, in seconds, before expiration time that the EvExpiring is fired.

Description

Sets the amount of time that should remain to a subscription before EvExpiring is called for that subscription.

This method affects only future subscriptions. This means that if there is an active subscription while this method is called, the EvExpiring event is called according to the previous threshold value.

If the expires value is smaller than or equal to the threshold, the EvExpiring event is never called, EvExpired is instead called directly.

See Also

Refresh (see page 543), Terminate (see page 545)

3.7.1.21.1.4 - ISipReferrerSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipReferrerMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipReferrerMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.21.1.5 - ISipReferrerSvc::Terminate Method

Terminates the subscription for that REFER.

C++

```
virtual mxt_result Terminate(IN mxt_opaque opqReferId, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqReferId	The opaque id corresponding to the REFER subscription to terminate.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership of this parameter is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership of this parameter is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

- resFE_INVALID_ARGUMENT: The id does not correspond to any subscription in this object.
- resFE_FAIL: The SUBSCRIBE request could not be sent.
- resS_OK: The SUBSCRIBE request has been successfully sent.

Description

This method sends a SUBSCRIBE request to immediately terminate the subscription created by the REFER it sent. It can be used when the application is no longer interested in the result of the referral process.

See Also

Refresh (See page 543)

3.7.1.22 - ISipRegistrationMgr Class

Class Hierarchy

```
ISipRegistrationMgr
```

C++

```
class ISipRegistrationMgr;
```

Description

The Registration Manager is the interface through which the Registration Service talks to the application. The ISipRegistrationSvc (see page 548) informs the application through this interface of the progress of the REGISTER requests it has sent and their expirations.

Location

SipUserAgent/ISipRegistrationMgr.h

See Also

ISipRegistrationSvc (see page 548)

Methods

Method	Description
◆ A EvExpired (see page 546)	Notifies the application that these contacts are expired.
◆ A EvExpiring (see page 547)	One or more contacts are about to expire.
◆ A EvFailure (see page 547)	A failure response (≥ 300) to a REGISTER request has been received by the ISipRegistrationSvc (see page 548).
◆ A EvProgress (see page 548)	A provisional response (< 200) to a REGISTER request has been received by the ISipRegistrationSvc (see page 548).
◆ A EvSuccess (see page 548)	A success response (2xx) to a REGISTER request has been received by the ISipRegistrationSvc (see page 548).

Legend

◆ A	Method
A	abstract

3.7.1.22.1 - Methods

3.7.1.22.1.1 - ISipRegistrationMgr::EvExpired Method

Notifies the application that these contacts are expired.

C++

```
virtual void EvExpired(IN ISipRegistrationSvc* pSvc, IN TO CSipHeader* pContacts) = 0;
```

Parameters

Parameters	Description
IN ISipRegistrationSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing the contacts.
IN TO CSipHeader* pContacts	A list of contacts that are expired. If there are many contacts, they will be chained in the CSipHeader (see page 239) object.

Description

Notifies the registration manager that the contacts in the parameter are expired. The manager should consider that these are no longer registered to the remote server. Usually, the manager should not let that event occur. If it wants to unregister contacts, it should not wait until their expiration but rather explicitly unregister them with the Remove method of the registration service interface. If the manager wants to keep the contacts registered to the remote server, it should register them again by calling the Add method of the Registration Service of this context. This event should only occur when the manager was unable to refresh or remove its registration due to some remote server error and when the contacts were received with a smaller expiration time than the threshold of the registration service.

See Also

ISipRegistrationSvc (see page 548), EvExpiring (see page 547)

3.7.1.22.1.2 - ISipRegistrationMgr::EvExpiring Method

One or more contacts are about to expire.

C++

```
virtual void EvExpiring(IN ISipRegistrationSvc* pSvc, IN TO CSipHeader* pContacts) = 0;
```

Parameters

Parameters	Description
IN ISipRegistrationSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing the contacts.
IN TO CSipHeader* pContacts	The list of contacts that are about to expire. If there are many contacts, they will be chained in the CSipHeader (see page 239) object.

Description

Notifies the registration manager that the contacts in the parameter are about to expire. If the manager wants to keep the contacts registered to the remote server, it should register them again by calling the Add method of the Registration Service of this context. If the manager fails to do so, EvExpired (see page 546) will be issued later, the amount of time being dependent on ISipRegistrationSvc::SetExpiringThreshold (see page 555)'s configuration.

See Also

ISipRegistrationSvc (see page 548), EvExpired (see page 546)

3.7.1.22.1.3 - ISipRegistrationMgr::EvFailure Method Updated behavior in 4.1.4

A failure response (≥ 300) to a REGISTER request has been received by the ISipRegistrationSvc (see page 548).

C++

```
virtual void EvFailure(IN ISipRegistrationSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRegistrationSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response packet.

Description

Notifies the registration manager that a failure response has been received for the last outgoing REGISTER request sent through the registration service.

After this failure response, the registration service acts as if the last REGISTER request was not sent.

Updated behaviour:

The registration service's behaviour regarding the reporting of this event has changed in the following ways with respect to SIP-UA 4.1.3.

Generally this event will be reported for failure responses but it may now also be reported for success responses. This situation will occur in one of the following cases:

- The last request sent tried to register new contacts and the 200 response received did not contain those "to be" registered contacts.
- The last request sent tried to remove some of the existing registered contacts and the 200 response contained one or more of those contacts.

- The last request sent tried to clear all registrations and the 200 response contained contacts.
- Any one of the 200 response contacts causes a parser error.

3.7.1.22.1.4 - ISipRegistrationMgr::EvProgress Method

A provisional response (< 200) to a REGISTER request has been received by the ISipRegistrationSvc ([see page 548](#)).

C++

```
virtual void EvProgress(IN ISipRegistrationSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRegistrationSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response packet.

Description

Notifies the registration manager that a provisional response has been received for the last outgoing REGISTER request sent through the registration service.

The state of the registration service does not change after the reception of this provisional response: it still waits for a final response.

3.7.1.22.1.5 - ISipRegistrationMgr::EvSuccess Method

A success response (2xx) to a REGISTER request has been received by the ISipRegistrationSvc ([see page 548](#)).

C++

```
virtual void EvSuccess(IN ISipRegistrationSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipRegistrationSvc* pSvc	The ISipRegistrationSvc (see page 548) that is managing this response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response packet.

Description

Notifies the registration manager that a success response has been received for the last outgoing REGISTER request sent through the registration service. Some validation is performed on the response packet to guarantee that when this event is reported the packet will be RFC3261 compliant. This means that after a call to ISipRegistrationSvc::Add ([see page 550](#)), the response packet will contain all sent contacts. After a call to ISipRegistrationSvc::Remove ([see page 554](#)), the response will not contain any of the sent contacts. After a call to ISipRegistrationSvc::Clear ([see page 552](#)), the response packet will not contain any contacts at all. And, after a call to ISipRegistrationSvc::GetList ([see page 553](#)), the packet may contain contacts. If any of the above conditions are not met, ISipRegistrationMgr::EvFailure ([see page 547](#)) will be reported instead.

After this success response, the registration service keeps track of registered contacts and warns the manager when they are about to expire and when they are expired.

See Also

ISipRegistrationMgr::EvFailure ([see page 547](#))

3.7.1.23 - ISipRegistrationSvc Class

Class Hierarchy



C++

```
class ISipRegistrationSvc : public IEComUnknown;
```

Description

The registration service offers the interface to manage registrations to a SIP registrar server. It notifies its manager when one or more registered contacts is about to expire or has expired.

This service monitors the expiration of the registrations that have been registered through it. It will notify its manager with ISipRegistrationMgr::EvExpiring (see page 547) a few seconds before a registration is about to expire, in order to allow to refresh it before it is discarded by the server. It will also report ISipRegistrationMgr::EvExpired (see page 546) when a registration becomes invalid on the server.

The number of seconds between ISipRegistrationMgr::EvExpiring (see page 547) and the real registration expiration can be configured through SetExpiringThreshold (see page 555).

This service will not initiate registration refreshes on its own. This must always be initiated by the user of this service.

This service supports first and third-party registrations. The From and To headers of the registrations sent by this service are the local address and remote address information configured in the ISipUserAgentSvc (see page 639) that is attached to the same ISipContext (see page 23) to which the registration service is attached. The registrar address is the Request-URI configured in the ISipUserAgentSvc (see page 639) that is attached to the same ISipContext (see page 23).

This service reports events to the application through the ISipRegistrationMgr (see page 546) interface.

The ISipRegistrationSvc is an ECOM object

The ISipRegistrationSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipRegistrationSvc

Interface Id: IID_ISipRegistrationSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Notes

There can be some confusion regarding the relation between the contact URL given in the call to ISipUserAgentSvc::AddLocalContact (see page 641) and the contacts that are added to registrations through this interface. The rContactUrl parameter of ISipUserAgentSvc::AddLocalContact (see page 641) is not used at all by the ISipRegistrationSvc, but it is still required to be the valid current request-URI of the local user.

Updated Behaviour:

The behaviour of the registration service has changed from MxSF v3.x.

- In v4.0, the service does not support values other than an integral number of seconds in the Expires header and in the expires parameter of a Contact header received from the registrar, as defined in RFC 3261.

If no valid expiration value is found in the packet received from the registrar, the default value of 3600 seconds (1 hour) is taken.

- In v4.0, the service does NOT generate EvExpiring if the registrar returns expiration times that are lower than the threshold. The next event that the manager receives for the corresponding registrations is EvExpired.
- In v4.0, if there are more than one of the same contact header found in the success response from the registrar, only the last found is used to set the expiration value of the registration.

Location

SipUserAgent/ISipRegistrationSvc.h

See Also

ISipRegistrationMgr (see page 546)

Methods

Method	Description
• A Add (see page 550)	Sends a REGISTER request with multiple Contact headers.
• A AddLocalRegistration (see page 551)	Sends a REGISTER request with multiple Contact headers whose local address field will be filled automatically based on the actual interface used to send the packet.
• A Clear (see page 552)	Removes all registration bindings from the registrar server.
• A GetContactMatchingType (see page 553)	Returns the contact's matching behaviour.
• A GetList (see page 553)	Requests the list of registered contacts from the registrar server.
• A GetRegId (see page 554)	
• A Remove (see page 554)	Removes one or more registration binding from the registrar server.
• A SetContactMatchingType (see page 555)	Sets the contact's matching behaviour.
• A SetExpiringThreshold (see page 555)	Sets the delay before expiration at which the EvExpiring event will fire.
• A SetIgnoreContactParamInSuccessResponse (see page 556)	
• A SetManager (see page 556)	Configures the manager of this service.
• A SetRegId (see page 557)	
• A UpdateContact (see page 557)	Updates the contacts expiration value for one contact managed by this service.

Legend

• A	Method
A	abstract

Enumerations

Enumeration	Description
EContactMatchingType (see page 558)	This defines the matching mode for contacts received from the server.
EThresholdType (see page 558)	Possible threshold types.

3.7.1.23.1 - Methods

3.7.1.23.1.1 - ISipRegistrationSvc::Add Method

Sends a REGISTER request with multiple Contact headers.

C++

```
virtual mxt_result Add(IN TO CSipHeader* pContacts, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN TO CSipHeader* pContacts	The contacts to register with the remote server. Adding multiple contacts in one call can be done by chaining multiple contact headers in the CSipHeader (see page 239) object. The raw header of all the headers in the chain is ignored and removed. Ownership of the object is always taken (even if this method fails).
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REGISTER request to be sent. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_ARGUMENT if the header pContacts is not a Contact header.

resFE_INVALID_STATE if there is no manager associated with this service, if a final response to a previous request has not been received, or if AddLocalRegistration (see page 551) was previously called on this service. No manager is associated with the service if

none was set with SetManager (see page 556) or if Clear (see page 552) was called on the Context.

resFE_SIPCORE_PACKET_BLOCKED One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method sends a REGISTER to create a binding on the SIP registrar server.

The REGISTER request contains all contact headers as provided in the pContacts parameter. No 'expires' parameter is added by this service, it is the user's responsibility to add one if needed.

If the request is successful (the remote server sent a 2xx response), this service monitors the registrations expiration and notifies its manager when it is about to expire or when it has actually expired. This service also updates all the registration expirations for all the contacts contained in the response and that have been registered through this service.

There can never be two REGISTER requests at the same time. That means that you have to wait for a FINAL response (through ISipRegistrationMgr::EvResponse) before calling this method again. If you call this method before receiving a final response, the method fails.

Note that the service does NOT generate any EvExpiring event if a registration value contained in the response is shorter than the value set with SetExpiringThreshold (see page 555). The next event generated for the corresponding contact is EvExpired if it has not been registered again.

Also note that if there are more than one of the same contact header found in the success response from the registrar, only the last found is used to set the expiration value of the registration.

This method is mutually exclusive with AddLocalRegistration (see page 551): only one of these can be called on any given instance of this service.

See Also

ISipRegistrationMgr::EvResponse

3.7.1.23.1.2 - ISipRegistrationSvc::AddLocalRegistration Method

Sends a REGISTER request with multiple Contact headers whose local address field will be filled automatically based on the actual interface used to send the packet.

C++

```
virtual mxt_result AddLocalRegistration(IN TO CSipHeader* pContacts, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN TO CSipHeader* pContacts	The contacts to register with the remote server. Adding multiple contacts in one call can be done by chaining multiple contact headers in the CSipHeader (see page 239) object. The raw header of all the headers in the chain is ignored and removed. The local address field of all contacts is filled automatically. Ownership of the object is always taken (even if this method fails).
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REGISTER request to be sent. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_ARGUMENT if the header pContacts is not a Contact header.

resFE_INVALID_STATE if there is no manager associated with this service, if a final response to a previous request has not been received, or if Add (see page 550) was already called on this service. No manager is associated with the service if none was set with SetManager (see page 556) or if Clear (see page 552) was called on the Context.

resFE_SIPCORE_PACKET_BLOCKED One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method notifies the service to take responsibility for correctly updating the contacts address field. It is otherwise identical to the Add (see page 550) method.

If the address of a contact was previously set, it is overriden.

AddLocalRegistration is mutually exclusive with Add (see page 550): only one of these methods can be called on any given instance of this service. This implies that creating locally-answered 3rd party registration is not possible with AddLocalRegistration: it is meant for true local registrations only. If both local and remote registrations must be handled simultaneously, two different instances of the service must be used and hence two different SIP contexts must be created. This is a change of behaviour from the previous stack version.

See Also

ISipRegistrationSvc::Add (see page 550)

3.7.1.23.1.3 - ISipRegistrationSvc::Clear Method

Removes all registration bindings from the registrar server.

C++

```
virtual mxt_result Clear(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REGISTER request to be sent. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager associated with this service or if a final response to a previous request has not been received. No manager is associated with the service if none was set with SetManager (see page 556) or if Clear was called on the Context.

resFE_SIPCORE_PACKET_BLOCKED One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method sends a REGISTER to remove ALL bindings from the registrar server. The REGISTER request contains a contact header with the wildcard value '*' and an Expire header with value 0, hence requesting the remote server to unbind the registration for every contact corresponding to this user.

If the request is successful (the remote server sent a 2xx response), this service stops monitoring the registration expirations for all the contacts registered through this service.

There can never be two REGISTER requests at the same time. That means that you have to wait for a FINAL response (through ISipRegistrationMgr::EvResponse) before calling this method again. If you call this method before receiving a final response, the method fails.

See Also

ISipRegistrationMgr::EvResponse

3.7.1.23.1.4 - ISipRegistrationSvc::GetContactMatchingType Method New in 4.1.8

Returns the contact's matching behaviour.

C++

```
virtual unsigned int GetContactMatchingType() const = 0;
```

Returns

An EContactMatchingType ([see page 558](#)) enumeration that specifies the current contact matching behaviour.

Description

Returns the matching behaviour for contacts received in responses from the server. See the EContactMatchingType ([see page 558](#)) enumeration documentation for details.

See Also

[SetContactMatchingType](#) ([see page 555](#)), [EContactMatchingType](#) ([see page 558](#))

3.7.1.23.1.5 - ISipRegistrationSvc::GetList Method

Requests the list of registered contacts from the registrar server.

C++

```
virtual mxt_result GetList(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REGISTER request to be sent. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager associated with this service or if a final response to a previous request was not received. No manager is associated with the service if none was set with [SetManager](#) ([see page 556](#)) or if [Clear](#) ([see page 552](#)) was called on the Context.

resFE_SIPCORE_PACKET_BLOCKED One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method sends a REGISTER request without any contact. The response from the server, if successful, should contain a list of the currently registered contacts.

If the request is successful (the remote server sent a 2xx response), this service updates all the registration expirations for all the contacts contained in the response and that have been registered through this service.

There can never be two REGISTER requests at the same time. That means that you have to wait for a FINAL response (through [ISipRegistrationMgr::EvResponse](#)) before calling this method again. If you call this method before receiving a final response, the method fails.

Note that NO [EvExpiring](#) event is generated by the service if a registration value contained in the response is shorter than the value set with [SetExpiringThreshold](#) ([see page 555](#)). The next event generated for the corresponding contact is [EvExpired](#) if it has not been registered again.

Also take note that if there are more than one of the same contact header found in the success response from the registrar, only the last found is used to set the expiration value of the registration.

See Also

ISipRegistrationMgr::EvResponse

3.7.1.23.1.6 - ISipRegistrationSvc::GetRegId Method *New in 4.1.6*

```
virtual unsigned int GetRegId() const = 0;
```

Returns

The reg-id parameter.

Description

Gets the reg-id to add in the contact headers of REGISTER requests when sent over an outbound connection.

See Also

ISipOutboundConnectionSvc (See page 102)

3.7.1.23.1.7 - ISipRegistrationSvc::Remove Method

Removes one or more registration binding from the registrar server.

C++

```
virtual mxt_result Remove(IN TO CSipHeader* pContacts, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN TO CSipHeader* pContacts	The contacts to remove from the remote server. Removing multiple contacts in one call can be done by chaining multiple contact headers in the CSipHeader (See page 239) object. The raw header of all the headers in the chain is ignored and removed. It must not be NULL. Ownership is always taken (even if this method fails).
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership of the object is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the REGISTER request to be sent. It can be NULL. Ownership of the object is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_ARGUMENT if the header pContacts is not a Contact header or if pContacts is NULL.

resFE_INVALID_STATE if there is no manager associated with this service or if a final response to a previous request has not been received. No manager is associated with the service if none was set with SetManager (See page 556) or if Clear (See page 552) was called on the Context.

resFE_SIPCORE_PACKET_BLOCKED One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request was correctly sent.

resFE_FAIL otherwise.

Description

This method sends a REGISTER to remove one or more bindings from the SIP registrar server.

The REGISTER request contains the contact headers as provided in the pContacts parameter. This method removes all the 'expires' parameter already in the contacts header's parameter and ensures that all contact headers include an 'expires=0' parameter.

If the request is successful (the remote server sent a 2xx response), this service stops monitoring the registration expirations for the specified contacts, if they previously had been registered through this service. This service also updates all the registration expirations for all the contacts contained in the response, that are not included in pContacts and that have been registered through this service.

There can never be two REGISTER requests at the same time. That means that you have to wait for a FINAL response (through

ISipRegistrationMgr::EvResponse) before calling this method again. If you call this method before receiving a final response, the method fails.

Note that the service does NOT generate any EvExpiring event if a registration value contained in the response is shorter than the value set with SetExpiringThreshold (see page 555). The next event generated for the corresponding contact is EvExpired if it has not been registered again.

Also take note that if there are more than one of the same contact header found in the success response from the registrar, only the last found is used to set the expiration value of the registration.

See Also

ISipRegistrationMgr::EvResponse

3.7.1.23.1.8 - ISipRegistrationSvc::SetContactMatchingType Method New in 4.1.8

Sets the contact's matching behaviour.

C++

```
virtual mxt_result SetContactMatchingType(IN unsigned int uContactMatchingType) = 0;
```

Parameters

Parameters	Description
IN unsigned int uContactMatchingType	One or more of the EContactMatchingType (see page 558) enumeration values to describe the expected behaviour.

Returns

- Success: The matching type has been modified.
- Failure: The requested matching type is invalid.

Description

Sets the matching behaviour for contacts received in responses from the server. See the EContactMatchingType (see page 558) enumeration documentation for details.

Backward Compatibility Note:

Applications that use the SetIgnoreContactParamInSuccessResponse (see page 556) are encouraged to use this new method instead. In order to obtain the same behaviour as for the:

- SetIgnoreContactParamInSuccessResponse (see page 556)(false) call, the application will use SetContactMatchingType(eMATCH_URI);
- SetIgnoreContactParamInSuccessResponse (see page 556)(true) call, the application will use SetContactMatchingType(eMATCH_USERNAME|eMATCH_HOST_PORT).

See Also

GetContactMatchingType (see page 553), EContactMatchingType (see page 558)

3.7.1.23.1.9 - ISipRegistrationSvc::SetExpiringThreshold Method Updated behavior in 4.1.4.4

Sets the delay before expiration at which the EvExpiring event will fire.

C++

```
virtual mxt_result SetExpiringThreshold(IN unsigned int uDelayInS, IN EThresholdType eThresholdType = eTHRESHOLD_FIXED) = 0;
```

Parameters

Parameters	Description
IN unsigned int uDelayInS	The delay between the EvExpiring and the expiration in seconds.

IN ETresholdType eThresholdType = eTHRESHOLD_FIXED	The threshold type to use. Default value is eTHRESHOLD_FIXED.
--	---

Returns

resFE_INVALID_STATE if there is no manager associated with this service. No manager is associated with the service if none was set with SetManager (see page 556) or if Clear (see page 552) was called on the Context.

resFE_FAIL if a processing error occurred.

resS_OK otherwise.

Description

This method configures the delay between when the ISipRegistrationMgr::EvExpiring (see page 547) is generated and the moment a binding expires. By default, this delay is 60 seconds.

This method affects only future registrations. This means that if there is an active registration while this method is called, the EvExpiring event is called according to the previous threshold value.

If the expires value is smaller than or equal to the threshold, the EvExpiring event is never called, EvExpired is instead called directly.

Updated Behaviour:

The behaviour of the registration service has changed from MxSF v3.x.

In v4.0, the service does NOT generate EvExpiring if the registrar returns expiration times that are lower than the threshold. The next event that the manager will receive for the corresponding registrations is EvExpired.

See Also

ISipRegistrationMgr::EvExpiring (see page 547)

3.7.1.23.1.10 - ISipRegistrationSvc::SetIgnoreContactParamInSuccessResponse Method

Deprecated in 4.1.8

`virtual void SetIgnoreContactParamInSuccessResponse(IN bool bIgnoreParam) = 0;`

Parameters

Parameters	Description
IN bool bIgnoreParam	True to ignore parameters added in the contact of the REGISTER's response. False otherwise.

Description

Parameters added in response to the contact are usually ignored as stated in section 19.1.4 of RFC3261. The exceptions are the maddr, transport, user, ttl and method parameters. If true is passed in parameter, no parameter will be validated in the contact returned in the REGISTER's response, even the parameters of the exception. If false, the parameter will be validated as stated in RFC3261.

This method is now deprecated in favor of the more versatile SetContactMatchingType (see page 555) and GetContactMatchingType (see page 553) methods.

See Also

SetContactMatchingType (see page 555), GetContactMatchingType (see page 553)

3.7.1.23.1.11 - ISipRegistrationSvc::SetManager Method

Configures the manager of this service.

C++

`virtual mxt_result SetManager(IN ISipRegistrationMgr* pMgr) = 0;`

Parameters

Parameters	Description
IN ISipRegistrationMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.23.1.12 - ISipRegistrationSvc::SetRegId Method New in 4.1.6

```
virtual void SetRegId(IN unsigned int uRegId) = 0;
```

Parameters

Parameters	Description
IN unsigned int uRegId	The reg-id param used when sending a REGISTER request over an outbound connection.

Description

Sets the reg-id to add in the contact headers of REGISTER requests when sent over an outbound connection. There is one reg-id per outbound persistent connection.

This value is only added to Contacts when the REGISTER is sent over an outbound persistent connection.

See Also

ISipOutboundConnectionSvc ([see page 102](#))

3.7.1.23.1.13 - ISipRegistrationSvc::UpdateContact Method

Updates the contacts expiration value for one contact managed by this service.

C++

```
virtual mxt_result UpdateContact(IN TO CSipHeader* pContactHdr, IN unsigned int uNewExpirations) = 0;
```

Parameters

Parameters	Description
IN TO CSipHeader* pContactHdr	The contact on which to perform the update. Note that only one contact can be given at a time. The provided CSipHeader (see page 239) MUST be of type eHDR_CONTACT. The ownership is taken.
IN unsigned int uNewExpirations	The new expiration value to assign to the given contact. If 0, uDEFAULT_EXPIRES_VALUE_IN_S is used.

Returns

- resS_OK when this method has been completed correctly.
- resFE_INVALID_ARGUMENT when rpContactHdr is NULL or is not of type eHDR_CONTACT.
- resFE_FAIL otherwise.

Description

This method updates the expiration value of the one contact given as a parameter provided that it is managed by this service. This method's behaviour regarding timers is the same as that of the Add ([see page 550](#)) method except that it will NEVER send a REGISTER request AND the provided contact MUST be already known by the service.

This method only supports integral number values of expires parameter, as defined in RFC 3261.

Timers associated with this contact will be restarted appropriately. If the new expiration value is below the expiring threshold, the EvExpiring event will not be reported.

The expected use case of this method is a contact expiration update after reception of a shortened reg-info event.

See Also

Add (see page 550), ISipRegistrationMgr::EvExpiring (see page 547), ISipRegistrationMgr::EvExpired (see page 546)

3.7.1.23.2 - Enumerations**3.7.1.23.2.1 - ISipRegistrationSvc::EContactMatchingType Enumeration**

This defines the matching mode for contacts received from the server.

C++

```
enum EContactMatchingType {
    eMATCH_URI = 0x0,
    eMATCH_HOST_PORT = 0x1,
    eMATCH_USERNAME = 0x2
};
```

Description

RFC 3261 states that contacts received from the registrar in a response MUST match the contacts sent in the REGISTER request as per Contact header matching rules.

However, this is sometimes too strict and the ISipRegisterSvc allows an application to loosen the matching rules by using one or more of the values in the EContactMatchingType enumeration.

This enumeration is a bitset. Values can be combined to achieve the required behaviour. Note that there is no point in combining any values with eMATCH_URI since it already checks everything.

Default Behaviour:

The matching default behaviour is eMATCH_USERNAME | eMATCH_HOST_PORT combination. This provides a looser matching than specified in RFC 3261.

Members

Members	Description
eMATCH_URI = 0x0	Match the complete contact's URI including any URI parameters, if any as per RFC3261, sections "10.2.4 Refreshing Bindings" and "19.1.4 URI Comparison". This type of matching, is incompatible with GRUU since the server MAY add GRUU specific information to the username.
eMATCH_HOST_PORT = 0x1	Match only the host port part of the contact's SIP URI.
eMATCH_USERNAME = 0x2	Match only the username of the contact's SIP URI. This type of matching, if used alone, will be incompatible with GRUU since the server MAY add GRUU specific information to the username.

3.7.1.23.2.2 - ISipRegistrationSvc::EThresholdType Enumeration

Possible threshold types.

C++

```
enum ETresholdType {
    eTHRESHOLD_FIXED,
    eTHRESHOLD_DYNAMIC
};
```

Description

These are the possible threshold types that can be used to calculate the time for EvExpiring.

Members

Members	Description
eTHRESHOLD_FIXED	Fixed value threshold.
eTHRESHOLD_DYNAMIC	Dynamic value threshold.

3.7.1.24 - ISipReliableProvisionalResponseMgr Class

Class Hierarchy

```
ISipReliableProvisionalResponseMgr
```

C++

```
class ISipReliableProvisionalResponseMgr;
```

Description

The reliable provisional response manager is the interface through which the corresponding service reports events to the application. The M5T stack informs the application, through this interface, of the reception of a reliable 1xx response or the reception of a PRACK to a reliable 1xx response. It is also through this interface that the application can have access to the various payloads carried with the SIP packets, which are usually updated SDP offers and answers.

Updated Behaviour:

The behaviour of the reliable provisional response service has changed from MxSF v3.x.

- An INVITE request requiring reliability in its provisional responses is NO LONGER reported to the manager. It is now the responsibility of the application to verify if the option-tag '100rel' is present in the Require header of the INVITE request. If so, the application MUST send non-100 provisional responses reliably. If the application does not want to do so, it MUST reject the INVITE with a 420 response where an Unsupported header contains the '100rel' option-tag. The application can use `ISipReliableProvisionalResponseSvc::IsReliabilitySupportedByPeer` (see page 564) and `ISipReliableProvisionalResponseSvc::IsReliabilityRequiredByPeer` (see page 563) to know if the peer UA supports or requires reliability in the provisional responses.

Location

SipUserAgent/ISipReliableProvisionalResponseMgr.h

See Also

`ISipReliableProvisionalResponseSvc` (see page 562)

Methods

Method	Description
◆ A EvInvalidPrack (see page 559)	Informs the application that an invalid PRACK request has been received.
◆ A EvPrackFailure (see page 560)	Indicates that a failure response to the last PRACK sent has been received.
◆ A EvPrackProgress (see page 560)	Indicates that a provisional response to the last sent PRACK has been received.
◆ A EvPrackSuccess (see page 561)	Indicates that a success response to the last PRACK sent has been received.
◆ A EvReliableProvisionalResponseTimeout (see page 561)	Indicates that no PRACK has been received and answered successfully (2xx) for the last provisional response sent.
◆ A EvReliableResponseReceived (see page 561)	Indicates that a reliable provisional response has been received.
◆ A EvResponseAcknowledged (see page 562)	Indicates that a PRACK for the last provisional response sent has been received.

Legend

◆	Method
◆ A	abstract

3.7.1.24.1 - Methods

3.7.1.24.1.1 - ISipReliableProvisionalResponseMgr::EvInvalidPrack Method

Informs the application that an invalid PRACK request has been received.

C++

```
virtual void EvInvalidPrack(IN ISipReliableProvisionalResponseSvc* pSvc, IN mxt_opaque opqApplicationData, IN
const CSipPacket& rPacket, IN mxt_result resReason) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the PRACK request.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rPacket	The invalid incoming PRACK.
IN mxt_result resReason	resFE_MISSING_HEADER: There is no RAck header in the PRACK request. The service has automatically replied with a 400 response with "Missing RAck header" as reason-phrase.
resFE_CALL_LEG_TRANSACTION_DOES_NOT_EXIST	The PRACK request does not match any reliable provisional response sent by the service. The service has automatically replied with a 481 response.

Description

An invalid PRACK request has been received. The request may be invalid because the service is not ready to receive it.

If the manager of the pSvc was not set, all the PRACK requests are responded with a 500 response.

3.7.1.24.1.2 - ISipReliableProvisionalResponseMgr::EvPrackFailure Method

Indicates that a failure response to the last PRACK sent has been received.

C++

```
virtual void EvPrackFailure(IN ISipReliableProvisionalResponseSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the PRACK request.
IN ISipClientEventControl* pClientEventCtrl	The interface through which access client event related functionality.
IN const CSipPacket& rPacket	The failure response to the PRACK request.

Description

This method is called when a failure response (>=300) to a PRACK is received. Either this method or EvPrackSuccess (see page 561) is called once for every PRACK request sent. When this method is called, the PRACK request was denied so the reliable provisional response process is not terminated. The application could send another PRACK request to try to finish the reliable provisional response process again.

Note that the stack reports a timeout by creating a 408 response and reporting it to the application (through this method if the request sent was sent by the ISipReliableProvisionalResponseSvc (see page 562)).

3.7.1.24.1.3 - ISipReliableProvisionalResponseMgr::EvPrackProgress Method

Indicates that a provisional response to the last sent PRACK has been received.

C++

```
virtual void EvPrackProgress(IN ISipReliableProvisionalResponseSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the PRACK request.
IN ISipClientEventControl* pClientEventCtrl	The interface through which access client event related functionality.
IN const CSipPacket& rPacket	The provisional response to the PRACK request.

Description

This method is called when a provisional response (1xx) to a PRACK request is received.

3.7.1.24.1.4 - ISipReliableProvisionalResponseMgr::EvPrackSuccess Method

Indicates that a success response to the last PRACK sent has been received.

C++

```
virtual void EvPrackSuccess(IN ISipReliableProvisionalResponseSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the PRACK request.
IN ISipClientEventControl* pClientEventCtrl	The interface through which access client event related functionality.
IN const CSipPacket& rPacket	The success response to the PRACK request.

Description

This method is called when a success response (2xx) to a PRACK is received. Either this method or EvPrackFailure (see page 560) is called once for every PRACK request sent. When this method is called, the processing of the reliable provisional response is terminated so the service is ready to receive a new reliable provisional response.

3.7.1.24.1.5 - ISipReliableProvisionalResponseMgr::EvReliableProvisionalResponseTimeout Method

Indicates that no PRACK has been received and answered successfully (2xx) for the last provisional response sent.

C++

```
virtual void EvReliableProvisionalResponseTimeout(IN ISipReliableProvisionalResponseSvc* pSvc, IN mxt_opaque opqTransaction) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the retransmission of the responses.
IN mxt_opaque opqTransaction	The opaque parameter associated with the server transaction.

Description

This method is called when no PRACK has arrived or was answered successfully (2xx) 64*T1 seconds after a reliable provisional response was sent. The application should consider that the reliable provisional response was not received by the remote party. RFC 3262 recommends that the INVITE transaction SHOULD be rejected with a 5xx response.

If a PRACK request has been received and not responded with a final response yet, the application MUST send a final response to the PRACK request. Until a PRACK request is responded successfully to, trying to send a reliable provisional response fails.

3.7.1.24.1.6 - ISipReliableProvisionalResponseMgr::EvReliableResponseReceived Method

Indicates that a reliable provisional response has been received.

C++

```
virtual void EvReliableResponseReceived(IN ISipReliableProvisionalResponseSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipClientEventControl* pClientEventCtrl	The interface through which access client event related functionality.
IN const CSipPacket& rPacket	The incoming reliable provisional response.
ISipReliableProvisionalResponseSvc	The service managing the reliable provisional responses to INVITE requests.

Description

This method is called upon receiving a valid reliable provisional response. When this event occurs, the application must call ISipReliableProvisionalResponseSvc::Prack (see page 565)() to acknowledge the response. Note that this method is not called to

retransmit the reliable response. Neither will it be called if the RSeq number in the response is more than one higher than the RSeq number in the last reliable response received.

See Also

ISipReliableProvisionalResponseSvc::Prack (see page 565)()

3.7.1.24.1.7 - ISipReliableProvisionalResponseMgr::EvResponseAcknowledged Method

Indicates that a PRACK for the last provisional response sent has been received.

C++

```
virtual void EvResponseAcknowledged(IN ISipReliableProvisionalResponseSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseSvc* pSvc	The service managing the PRACK request.
IN ISipServerEventControl* pServerEventCtrl	The interface through which the application must reply at least a final response.
IN const CSipPacket& rPacket	The incoming PRACK.

Description

This method is called when a PRACK is received and its RAck header matches the last reliable provisional response that was sent. The application must use ISipServerEventControl::SendResponse (see page 76)() to send a final response to this PRACK.

This method is called every time a PRACK request is received. It can be called more than once per reliable provisional response sent. The application should accept one PRACK request with a 200 OK response and reject the others with a proper error status code. When a failure response is sent to all the PRACK received, the application should be prepared to receive new PRACK requests for the reliable provisional response. The reliable provisional response processing is not terminated until a 2xx response is sent for a PRACK received for it.

See Also

ISipServerEventControl::SendResponse (see page 76)

3.7.1.25 - ISipReliableProvisionalResponseSvc Class

Class Hierarchy



C++

```
class ISipReliableProvisionalResponseSvc : public IEComUnknown;
```

Description

The reliable provisional response service allows the application to support the 100rel extension as defined in RFC 3262. This service is responsible for sending and receiving reliable provisional responses for INVITE transaction. When sending a reliable provisional response, it automatically takes care of retransmitting the response until a corresponding PRACK is received and responded 2xx or a timeout occurs. When receiving a reliable provisional response, it notifies the application of only one of the retransmitted responses and silently catches the retransmissions. This service also handles the extension tag '100rel' that needs to be put in Supported or Require header fields.

This service permits to send or receive reliable provisional responses to an INVITE request one at a time. This means that the application using this service as UAS can only send one reliable provisional response at a time, and also that an application using this service as UAC can only receive one reliable provisional response at a time (the others are ignored). The reliable provisional response process is considered over when a 2xx response was sent or received for a PRACK request corresponding to the provisional response.

RFC 3262 states that a 2xx response MUST NOT be sent until all reliable provisional responses are acknowledged if any of the reliable provisional responses contained a session description. To be compliant with this condition, the reliable provisional service delays a 2xx response to an INVITE if there is still an unacknowledged reliable provisional response until a corresponding PRACK is received and responded successfully or the reliable provisional response times out.

Note that the usage of reliable provisional responses has implications where SDP offer and answer can appear. In particular, RFC 3264

states that when a SDP offer appears in an INVITE request, the answer may appear in any reliable response to the INVITE. When the 100rel extension is used, the responses sent through the reliable provisional response service are considered reliable responses and can hence contain the answer as well as the 2xx response sent by the session service. Also, if the INVITE did not contain an offer, RFC 3261 mandates that the offer MUST be sent in the first reliable response to that INVITE.

For more information on SDP offers and reliable provisional responses, please see section 5 in RFC 3262.

This service generates events to the application through the **ISipReliableProvisionalResponseMgr** (see page 559) interface.

The **ISipReliableProvisionalResponseSvc** is an ECOM object

The **ISipReliableProvisionalResponseSvc** is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipReliableProvisionalResponseSvc

Interface Id: IID_ISipReliableProvisionalResponseSvc

A user can query the **ISipContext** (see page 23) to which this service is attached by calling **QueryIf** on it. It can also directly access all other services attached to the **ISipContext** (see page 23) through the same mean.

Location

SipUserAgent/ISipReliableProvisionalResponseSvc.h

See Also

ISipReliableProvisionalResponseMgr (see page 559)

Methods

Method	Description
• A IsReliabilityRequiredByPeer (see page 563)	Notifies whether or not the remote endpoint insists on reliable delivery of provisional response.
• A IsReliabilitySupportedByPeer (see page 564)	Indicates whether or not the remote endpoint supports the 100rel extension.
• A LocalRequestsRequireReliability (see page 564)	Configures whether or not reliable provisional responses are required for local INVITE requests.
• A MakeReliableServerEventControl (see page 564)	Gives an ISipServerEventControl (see page 75) interface through which provisional responses are sent reliably.
• A Prack (see page 565)	Sends a PRACK request to acknowledge a reliable provisional response received.
• A SetManager (see page 566)	Configures the manager associated with this service.

Legend

•	Method
A	abstract

3.7.1.25.1 - Methods

3.7.1.25.1.1 - **ISipReliableProvisionalResponseSvc::IsReliabilityRequiredByPeer** Method

Notifies whether or not the remote endpoint insists on reliable delivery of provisional response.

C++

```
virtual bool IsReliabilityRequiredByPeer() = 0;
```

Returns

true when the remote endpoint insists on reliable delivery of provisional response.

false otherwise.

Description

Returns true if the last INVITE received in the parent context while there was no other INVITE pending contained a **Require** header with the '100rel' option-tag. If no such header was present, or if no INVITE has been received yet, it returns false.

When true is returned, the application MUST send non-100 provisional responses reliably. The application must call

MakeReliableServerEventControl (see page 564) to get a reliable ISipServerEventControl (see page 75) interface through which non-100 provisional responses are sent reliably.

3.7.1.25.1.2 - ISipReliableProvisionalResponseSvc::IsReliabilitySupportedByPeer Method

Indicates whether or not the remote endpoint supports the 100rel extension.

C++

```
virtual bool IsReliabilitySupportedByPeer() = 0;
```

Returns

true when the remote endpoint supports the 100rel extension
false otherwise.

Description

Returns true if the last INVITE received in the parent context while there was no other pending INVITE request contained either a Require or a Supported with the '100rel' option-tag. If no such header was present, or if no INVITE has been received yet, it returns false.

When false is returned, the peer UA does not support reliability in the provisional responses so reliable provisional responses MUST NOT be sent.

3.7.1.25.1.3 - ISipReliableProvisionalResponseSvc::LocalRequestsRequireReliability Method

Configures whether or not reliable provisional responses are required for local INVITE requests.

C++

```
virtual mxt_result LocalRequestsRequireReliability(IN bool bRequire) = 0;
```

Parameters

Parameters	Description
IN bool bRequire	Requires the usage of reliable provisional responses when true. Otherwise, lets the remote endpoint decide whether or not reliable responses are used.

Returns

resFE_INVALID_STATE if this service has no manager. This could happen when the manager was not set or when Clear was called on the context.

resS_OK if the method succeeded

Description

Use this method to require usage of reliable provisional response from the remote party.

This method has an impact only when this service is acting as a UAC. When setting the bRequire to true it puts a "Require: 100rel" header in every INVITE request sent. The remote party must then only send reliable provisional responses (except for the 100 Trying which cannot be sent reliably since it is processed hop by hop) or reject the call. If bRequire is set to false, the "Supported: 100rel" header is put in every INVITE instead. The remote party can then decide whether or not to send its provisional responses reliably.

Reliable provisional responses are received through the ISipReliableProvisionalResponseMgr::EvReliableResponseReceived (see page 561) method.

When the service is created, the default value is false.

See Also

ISipReliableProvisionalResponseMgr (see page 559)

3.7.1.25.1.4 - ISipReliableProvisionalResponseSvc::MakeReliableServerEventControl Method

Gives an ISipServerEventControl (see page 75) interface through which provisional responses are sent reliably.

C++

```
virtual mxt_result MakeReliableServerEventControl(IN ISipServerEventControl* pServerEventCtrl, OUT
ISipServerEventControl*& rpRelServerEventCtrl) = 0;
```

Parameters

Parameters	Description
IN ISipServerEventControl* pServerEventCtrl	The ISipServerEventControl (see page 75) interface given in the event signaling the reception of the INVITE request (normally ISipSessionMgr::EvInvited (see page 573) or ISipSessionMgr::EvReInvited (see page 575)). This object is used by rpRelServerEventCtrl to send requests and to manage opaque values. It MUST NOT be NULL.
OUT ISipServerEventControl*& rpRelServerEventCtrl	The reliable ISipServerEventControl (see page 75) interface created by using pServerEventCtrl. Reliable provisional responses are sent through rpRelServerEventCtrl. rpRelServerEventCtrl must be used to send all types of responses (it must replace pServerEventCtrl). The opaque value of this ISipServerEventControl (see page 75) object is the same as pServerEventCtrl. It is NULL if there was an error while calling this method.
IMPORTANT	If this method succeeds, this interface must no longer be used. The application should release it and use rpRelServerEventCtrl instead. Doing otherwise may cause problems in the reliable provisional response service.

Returns

resFE_INVALID_STATE if this service has no manager. This could happen when the manager was not set or when Clear was called on the context. OR if there is no INVITE request pending that supports reliability of provisional responses.

resFE_INVALID_ARGUMENT if pServerEventCtrl is not the interface managing the INVITE request in which a reliable provisional response can be sent.

resS_OK if all is correct, in this case, rpRelServerEventCtrl takes the value of a reliable ISipServerEventControl (see page 75) interface.

Description

This method gives an ISipServerEventControl (see page 75) interface that must be used to send all the remaining responses to the INVITE request. Non-100 provisional responses are automatically sent reliably with this ISipServerEventControl (see page 75) interface.

To succeed, this method needs the ISipServerEventControl (see page 75) given in the event signaling the reception of the INVITE request.

Note that the application must not send reliable provisional responses if it is not supported by the peer user agent. To know it, the application should call IsReliabilitySupportedByPeer (see page 564). If the application calls this method when reliability is not supported by the peer user agent, it fails.

See Also

IsReliabilitySupportedByPeer (see page 564), IsReliabilityRequiredByPeer (see page 563)

3.7.1.25.1.5 - ISipReliableProvisionalResponseSvc::Prack Method

Sends a PRACK request to acknowledge a reliable provisional response received.

C++

```
virtual mxt_result Prack(IN mxt_opaque transactionOpaque, IN TO CHeaderList* pExtraHeaders, IN TO
CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque transactionOpaque	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. Adding a RAck header in this header list is useless since this method replaces it with the RAck header it generates. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Message-body to include with the PRACK. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE: When the manager of this service has not been set or Clear was called on the context.
 resFE_INVALID_STATE: If no reliable response has been received.
 resFE_FAIL: When a PRACK request has already been sent for the reliable provisional response.
 resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
 resS_OK: If the request has been sent.

Description

Upon reception of a reliable provisional response (through the ISipReliableProvisionalResponseMgr::EvReliableResponseReceived (see page 561)() event), a PRACK request MUST be sent to acknowledge reception of the response. This method should be used to send this PRACK. It returns an error if there is no reliable provisional response to acknowledge or if there was already a PRACK request sent. This method automatically takes care of adding the RAck header that is mandatory in PRACK requests.

RFC 3262 describes when a content should accompany the PRACK request. When needed, such a content can be set through the pMessageBody parameter.

Extra headers can be added to the PRACK request through the pExtraHeaders. If pExtraHeaders already contains a RAck header, the header is removed and replaced by the RAck header that the service has generated.

See Also

ISipReliableProvisionalResponseMgr (see page 559)

3.7.1.25.1.6 - ISipReliableProvisionalResponseSvc::SetManager Method

Configures the manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipReliableProvisionalResponseMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipReliableProvisionalResponseMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.
 resS_OK otherwise.

Description

Configures the manager that will receive the events generated by this service.
 Note that a manager MUST be associated with this service before it is used.

3.7.1.26 - ISipReplacesMgr Class**Class Hierarchy**

```
ISipReplacesMgr
```

C++

```
class ISipReplacesMgr;
```

Description

The replaces manager is the interface through which the replaces service reports events to the application. The replaces service informs the application through this interface that an INVITE request containing a Replaces headers has been received, whether or not it contains an error and whether or not it corresponds to an existing dialog.

Location

SipUserAgent/ISipReplacesMgr.h

See Also

ISipReplacesSvc (see page 568)

Methods

Method	Description
◆ A EvInvalidReplaces (see page 567)	Reports that a SIP INVITE request containing a Replaces header has been received, which was wrong or did not correspond to a single existing dialog.
◆ A EvReplaces (see page 567)	Reports that a SIP INVITE request containing a Replaces header was received and that a corresponding dialog was found.

Legend

◆	Method
A	abstract

3.7.1.26.1 - Methods**3.7.1.26.1.1 - ISipReplacesMgr::EvInvalidReplaces Method**

Reports that a SIP INVITE request containing a Replaces header has been received, which was wrong or did not correspond to a single existing dialog.

C++

```
virtual void EvInvalidReplaces(IN ISipReplacesSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest, IN mxt_result res) = 0;
```

Parameters

Parameters	Description
IN ISipReplacesSvc* pSvc	The ISipReplacesSvc (see page 568) managing the incoming request.
IN mxt_opaque opqApplicationData	The opaque data set by the application in a previous event for this same received request. When this is the first event for this request, this parameter is 0.
IN const CSipPacket& rRequest	The SIP request.
IN mxt_result res	<p>The error result:</p> <ul style="list-style-type: none"> resFE_FAIL when no dialog was found. A "481 Call/Transaction Does Not Exist" response has been automatically sent. resFE_INVALID_ARGUMENT when there is an error in the Replaces header or there is more than one Replaces header. A "400 Bad Request" response has been automatically sent. resFE_INVALID_STATE when the dialog matcher list has not been set beforehand or if the manager was not set. A "500 Server Internal Error" response has been automatically sent.

Description

Informs the manager that a SIP INVITE request containing a Replaces header has been received, which was wrong or did not correspond to a single existing dialog.

3.7.1.26.1.2 - ISipReplacesMgr::EvReplaces Method

Reports that a SIP INVITE request containing a Replaces header was received and that a corresponding dialog was found.

C++

```
virtual void EvReplaces(IN ISipReplacesSvc* pSvc, IN ISipUserAgentSvc* pReplacedUaSvc, IN bool bEarlyOnly, IN
const CSipPacket& rRequest, INOUT mxt_opaque& rApplicationData) = 0;
```

Parameters

Parameters	Description
IN ISipReplacesSvc* pSvc	The ISipReplacesSvc (see page 568) managing the incoming request.
IN ISipUserAgentSvc* pReplacedUaSvc	The User Agent Service of the dialog corresponding to the parameters in the Replaces header. It cannot be NULL.
IN bool bEarlyOnly	true if the early-only parameter was present in the Replaces header, false otherwise.
IN const CSipPacket& rRequest	The SIP request.
INOUT mxt_opaque& rApplicationData	Application data opaque to the service. This parameter is an INOUT parameter. It is used to correlate the events reported by multiple services for a unique received request. If the application already received an event for that request through another manager interface, rApplicationData equals the value stored in it by the application. Otherwise, if it is the first event reported for this received request, rApplicationData is set to 0. The value of rApplicationData when EvReplaces returns is accessible through the GetOpaque method of the ISipServerEventControl (see page 75) interface that accompanies the request when the owner service issues its event. This opaque data should be used to store the state indicating that a valid Replace was received and be able to act accordingly when the session manager receives the EvInvited event. Note that this event cannot be processed asynchronously since the opaque application data is passed by value to the ISipServerEventControl (see page 75) interface.

Description

Notifies the manager that a SIP INVITE request containing a Replaces header has been received and that a corresponding dialog has been found.

3.7.1.27 - ISipReplacesSvc Class**Class Hierarchy****C++**

```
class ISipReplacesSvc : public IEComUnknown;
```

Description

The replaces service is used to determine cases where a dialog can be replaced by another. It generates events to the application through the ISipReplacesMgr (see page 566) interface.

Its purpose is to analyse incoming INVITE requests and to check whether or not they contain a Replaces header. In the affirmative, if a single existing dialog corresponds to the header's parameters, it calls an EvReplaces event. If no dialog, or more than one dialog, corresponds to the header's parameters, a 481 Call Does Not Exist response is sent back. If the header is wrong, a 400 Bad Request response is sent back. In both error cases, an EvReplacesWithError event is returned to the application. Such an event is also returned when the dialog matcher list has not been set beforehand.

See RFC 3891 for more details on Replaces headers mechanism.

The ISipReplacesSvc is an ECOM object

The ISipReplacesSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipReplacesSvc

Interface Id: IID_ISipReplacesSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipReplacesSvc.h

Methods

Method	Description
• A SetManager (see page 569)	Configures the event manager associated with this service.

Legend

•	Method
A	abstract

3.7.1.27.1 - Methods

3.7.1.27.1.1 - ISipReplacesSvc::SetManager Method

Configures the event manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipReplacesMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipReplacesMgr* pMgr	The event manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT: pMgr is NULL.

resS_OK: Otherwise.

Description

Configures the event manager that will receive the events generated by this service.

Note that an event manager MUST be associated with this service before it is used.

3.7.1.28 - ISipSessionMgr Class

Class Hierarchy

ISipSessionMgr

C++

```
class ISipSessionMgr;
```

Description

The session manager is the interface through which the session service reports events to the application. The SIP Stack informs the application through this interface of the progress for INVITE requests, that an INVITE request has been received, that an ACK for a successful final response (2xx) to an INVITE has been received, and that the session has been terminated by the remote user.

It is also through this interface that the SIP Stack asks the application to create a new context when an outgoing INVITE response requires such action.

Updated Behaviour:

Some behaviours of the session service manager have changed from M5T SIP UA 3.x versions.

- EvTerminated (see page 576) is not called for the reception of a CANCEL request. It is only called for the reception of a BYE request. The reception of a CANCEL request that triggers the termination of a session is signaled through the new event EvInviteCancelled (see page 572).
- EvTimeout no longer exists. Failure to receive a final response to

an outgoing INVITE and BYE request are reported through the event `EvFailure` (see page 571) and `EvTerminationFailure` (see page 576) respectively. Failure to receive an ACK request following the sending of a final positive response is reported through a new event called `EvInviteSuccessResponseTimeout` (see page 573).

Location

`SipUserAgent/ISipSessionMgr.h`

See Also

`ISipSessionSvc` (see page 578)

Methods

Method	Description
◆ A <code>EvAcknowledged</code> (see page 570)	Notifies the application that an ACK request has been received for its successful final response (2xx).
◆ A <code>EvFailure</code> (see page 571)	Notifies the application that a final negative response has been received for the last outgoing INVITE request.
◆ A <code>EvInvalidInvite</code> (see page 571)	Warns the application that an INVITE or BYE request has been declined with a final negative response.
◆ A <code>EvInviteCancelled</code> (see page 572)	Notifies the application that the last INVITE request it received has been cancelled.
◆ A <code>EvInvited</code> (see page 573)	Notifies the application that an INVITE request has been received.
◆ A <code>EvInviteSuccessResponseTimeout</code> (see page 573)	Notifies the application that an INVITE server transaction timed out.
◆ A <code>EvNewSessionNeededForOriginalInviteResponse</code> (see page 574)	Notifies the application that a new context is needed to handle the given original INVITE response.
◆ A <code>EvProgress</code> (see page 574)	Notifies the application that a 1xx response has been received for the last outgoing INVITE request.
◆ A <code>EvReInviteCancelled</code> (see page 574)	Notifies the application that the last reINVITE request it received has been cancelled.
◆ A <code>EvReInvited</code> (see page 575)	Notifies the application that a re-INVITE request has been received.
◆ A <code>EvSuccess</code> (see page 575)	Notifies the application that a 2xx response has been received for the last outgoing INVITE request.
◆ A <code>EvTerminated</code> (see page 576)	Notifies the application that the remote party wants to terminate the session by using a BYE request.
◆ A <code>EvTerminationFailure</code> (see page 576)	Notifies the application that a final negative response has been received for the last outgoing BYE request.
◆ A <code>EvTerminationProgress</code> (see page 577)	Notifies the application that a provisionnal response has been received for the last outgoing BYE request.
◆ A <code>EvTerminationSuccess</code> (see page 577)	Notifies the application that a 2xx response has been received for the last outgoing BYE request.

Legend

◆	Method
A	abstract

3.7.1.28.1 - Methods

3.7.1.28.1.1 - `ISipSessionMgr::EvAcknowledged` Method

Notifies the application that an ACK request has been received for its successful final response (2xx).

C++

```
virtual void EvAcknowledged(IN ISipSessionSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The <code>ISipSessionSvc</code> (see page 578) managing the incoming request.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request or the corresponding INVITE request.
IN const CSipPacket& rRequest	The SIP request.

Description

Informs the application that the ACK for a successful final response (2xx) sent on this session service has been received. ACK for failure final response are automatically managed by the SIP Stack and are NOT notified through this method.

Only the ACK requests to 2xx responses are reported to the application because they can contain a content in the payload or they are end-to-end instead of hop-by-hop.

See Also

ISipSessionSvc (see page 578), EvInvited (see page 573), EvReInvited (see page 575)

3.7.1.28.1.2 - ISipSessionMgr::EvFailure Method

Notifies the application that a final negative response has been received for the last outgoing INVITE request.

C++

```
virtual void EvFailure(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rRequest	The SIP request.

Description

Notifies the application that a final negative response has been received. Before reporting the event, the SIP Stack automatically sends the ACK request.

This event can be called on a new context created because of a call to the event EvNewSessionNeededForOriginalInviteResponse (see page 574).

The INVITE request associated with this event has been sent on this context or on the context that reported the event EvNewSessionNeededForOriginalInviteResponse (see page 574).

See Also

ISipSessionSvc (see page 578)

3.7.1.28.1.3 - ISipSessionMgr::EvInvalidInvite Method

Warns the application that an INVITE or BYE request has been declined with a final negative response.

C++

```
virtual void EvInvalidInvite(IN ISipSessionSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest, IN mxt_result resReason) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The service managing this request.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rRequest	The request automatically declined by the SIP Stack.

<pre>IN mxt_result resReason</pre>	<ul style="list-style-type: none"> • resFE_INVALID_CONTACT_HEADER if the received INVITE request did not contain a Contact header, contained an invalid Contact header, or contained more than one Contact headers. In these cases, an automatic 400 "Missing, Erroneous or Multiple Contact header field(s)" response has been sent. • resFE_NO_ISIPUSERAGENTSVC_ATTACHED: <ul style="list-style-type: none"> • An INVITE or BYE request has been received while there was no ISipUserAgentSvc (see page 639) attached to the context. In this case, an automatic 500 response has been sent. • An INVITE or BYE request has been received while there was no manager set on this object. In this case, an automatic 500 response has been sent. • resFE_CALL_LEG_TRANSACTION_DOES_NOT_EXIST: If a BYE request has been received while there was no dialog established or an INVITE request has been received after the dialog has been terminated. In this case, an automatic 481 response has been sent. • resFE_REQUEST_PENDING: <ul style="list-style-type: none"> • An INVITE or BYE request has been received while the same request is proceeding as a UAC. In this case, an automatic 491 response has been sent. • An INVITE or BYE request has been received while the same request is proceeding as a UAS. In this case, an automatic 500 response has been sent. • An INVITE has been received while a BYE is processing (either as a UAC or UAS). In this case, an automatic 500 response has been sent.
------------------------------------	---

Description

This method is called when an INVITE or BYE request is received while the service cannot accept it.

Please note that if the manager of the service is not set, all the requests are responded with a 500 response.

Warning

The name of this method can be misleading. Indeed this event reports both invalid INVITE and invalid BYE requests.

3.7.1.28.1.4 - ISipSessionMgr::EvInviteCancelled Method

Notifies the application that the last INVITE request it received has been cancelled.

C++

```
virtual void EvInviteCancelled(IN ISipSessionSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The CANCEL SIP request.

Description

Notifies the application that the last INVITE request it received has been cancelled. Before issuing this method, the SIP Stack automatically sends a 200 "OK" in response to the CANCEL request.

The application MUST send a 487 "Request Cancelled" to the INVITE request by using the pServerEventCtrl given in parameter of this method.

See Also

ISessionSvcMgr, EvInvited (see page 573)

3.7.1.28.1.5 - ISipSessionMgr::EvInvited Method

Notifies the application that an INVITE request has been received.

C++

```
virtual void EvInvited(IN ISipSessionSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The SIP request.

Description

Notifies the application that a remote user is trying to establish a dialog with the local user by sending an INVITE request. The application should send responses through the ISipServerEventControl::SendResponse (see page 76) method. If the application has not sent any response within 200 ms, the SIP Stack sends a 100 "Trying" response in order to stop retransmission. If the application releases the context without sending any final response, a 487 "Request Terminated" is sent. If the application sends a successful final response (2xx), the SIP Stack issues the EvAcknowledged (see page 570) event upon receiving the ACK request from the remote server.

See Also

ISipSessionSvc (see page 578), ISipServerEventControl (see page 75), EvAcknowledged (see page 570)

3.7.1.28.1.6 - ISipSessionMgr::EvInviteSuccessResponseTimeout Method

Notifies the application that an INVITE server transaction timed out.

C++

```
virtual void EvInviteSuccessResponseTimeout(IN ISipSessionSvc* pSvc, IN mxt_opaque opq, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN mxt_opaque opq	The opaque parameter associated with this server transaction.
IN const CSipPacket& rResponse	The SIP 2xx response.

Description

Informs the application that a timeout occurred when sending a final 2xx response to an INVITE and failing to receive an ACK. The application can then call the Bye method on the session service.

See Also

ISipSessionSvc::Bye (see page 580)

3.7.1.28.1.7 - **ISipSessionMgr::EvNewSessionNeededForOriginalInviteResponse** Method

Notifies the application that a new context is needed to handle the given original INVITE response.

C++

```
virtual void EvNewSessionNeededForOriginalInviteResponse(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The session service.
IN ISipClientEventControl* pClientEventCtrl	The client event control.
IN const CSipPacket& rPacket	Do not use. This parameter is always NULL. The response that requires a new context.

Description

Informs the application that a new context is needed to handle the given original INVITE response. This event is reported to the application when a response to an initial INVITE has a To tag different from the To tag of every other response received before for that INVITE.

This can be the case when an outgoing INVITE request is forked by a Proxy. The first response received for the original INVITE request is associated with the session created to send the INVITE. All other INVITE responses that have a different To tag need a new session to be handled properly.

The application has to create a new session, attach services to it, and give it back to the session service that issued the event. The application SHOULD attach all the same services that are attached to the original INVITE context.

See Also

[ISipSessionSvc::HandleOriginalInviteResponseNewSession](#) (see page 580), [ISipContext::AttachService](#) (see page 25)

3.7.1.28.1.8 - **ISipSessionMgr::EvProgress** Method

Notifies the application that a 1xx response has been received for the last outgoing INVITE request.

C++

```
virtual void EvProgress(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rRequest	The SIP request.

Description

Notifies the application that a provisional response has been received. It could be any response with a 1xx code. 100 Trying and 180 Ringing responses are reported through this event.

This event can be called on a new context created because of a call to the event [EvNewSessionNeededForOriginalInviteResponse](#) (see page 574).

The INVITE request associated with this event has been sent on this context or on the context that reported the event [EvNewSessionNeededForOriginalInviteResponse](#) (see page 574).

See Also

[ISipSessionSvc](#) (see page 578)

3.7.1.28.1.9 - **ISipSessionMgr::EvReInviteCancelled** Method

Notifies the application that the last reINVITE request it received has been cancelled.

C++

```
virtual void EvReInviteCancelled(IN ISipSessionSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The CANCEL SIP request.

Description

Notifies the application that the last reINVITE request it received has been cancelled. Before issuing this method, the SIP Stack automatically sends a 200 "OK" in response to the CANCEL request.

The application MUST send a 487 "Request Cancelled" to the INVITE request by using the pServerEventCtrl given in parameter of this method.

The application should stop every processing associated with this reINVITE and fall back to the previously agreed media description (SDP).

See Also

ISessionMgr, EvReInvited (see page 575)

3.7.1.28.1.10 - ISipSessionMgr::EvReInvited Method

Notifies the application that a re-INVITE request has been received.

C++

```
virtual void EvReInvited(IN ISipSessionSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The SIP request.

Description

Informs the application that the already connected remote user is sending an INVITE request in order to change the agreed session description or simply refresh the current session (as part of a keep alive mechanism). The application should send responses through the ISipServerEventControl::SendResponse (see page 76) method. If the application has not sent any response within 200 ms, the SIP Stack sends a 100 "Trying" response in order to stop retransmission.

If the application sends a successful final response (2xx), the SIP Stack issues the EvAcknowledged (see page 570) event upon receiving the ACK request from the remote server.

See Also

ISipSessionSvc (see page 578), ISipServerEventControl (see page 75), EvAcknowledged (see page 570)

3.7.1.28.1.11 - ISipSessionMgr::EvSuccess Method

Notifies the application that a 2xx response has been received for the last outgoing INVITE request.

C++

```
virtual void EvSuccess(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rRequest	The SIP request.

Description

Notifies the application that a 2xx response has been received.

This event can be called on a new context created because of a call to the event `EvNewSessionNeededForOriginalInviteResponse` (see page 574).

The INVITE request associated with this event has been sent on this context or on the context that reported the event `EvNewSessionNeededForOriginalInviteResponse` (see page 574).

The application must send the ACK through the session service `Ack` method.

See Also

`ISipSessionSvc` (see page 578)

3.7.1.28.1.12 - `ISipSessionMgr::EvTerminated` Method

Notifies the application that the remote party wants to terminate the session by using a BYE request.

C++

```
virtual void EvTerminated(IN ISipSessionSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the incoming request.
IN ISipServerEventControl* pServerEventCtrl	Interface to the server event control for this server transaction.
IN const CSipPacket& rRequest	The SIP request.

Description

Informs the application that it should no longer process the session associated with this context. This method is triggered by the reception of a BYE request.

The application has to respond a final response to any pending BYE request. It MUST also respond a 487 "Request Terminated" to any pending server INVITE transaction. A server transaction is considered to be pending if no final response (>= 2xx) has been sent yet.

See Also

`ISipSessionSvc` (see page 578).

3.7.1.28.1.13 - `ISipSessionMgr::EvTerminationFailure` Method

Notifies the application that a final negative response has been received for the last outgoing BYE request.

C++

```
virtual void EvTerminationFailure(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The ISipSessionSvc (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rResponse	The SIP request.

Description

Notifies the application that a >= 3xx response has been received for the last BYE sent.

The application should try to fix the situation and reissue the BYE request. If it cannot fix the situation, it should consider the session terminated as if `EvTerminationSuccess` (see page 577) was received.

See Also

`ISipSessionSvc` (see page 578)

3.7.1.28.1.14 - `ISipSessionMgr::EvTerminationProgress` Method

Notifies the application that a provisionnal response has been received for the last outgoing BYE request.

C++

```
virtual void EvTerminationProgress(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The <code>ISipSessionSvc</code> (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rRequest	The SIP request.

Description

Notifies the application that a provisionnal response has been received for the last BYE sent.

Note that although it is recommended to not send provisional responses (1xx) to requests other than INVITE, it is possible that this event may be called. The application should not release the context until reception of a final response.

See Also

`ISipSessionSvc` (see page 578)

3.7.1.28.1.15 - `ISipSessionMgr::EvTerminationSuccess` Method

Notifies the application that a 2xx response has been received for the last outgoing BYE request.

C++

```
virtual void EvTerminationSuccess(IN ISipSessionSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSessionSvc* pSvc	The <code>ISipSessionSvc</code> (see page 578) managing the outgoing request.
IN ISipClientEventControl* pClientEventCtrl	Interface to the client event control for this client transaction.
rRequest	The SIP request.

Description

Notifies the application that a 2xx response has been received for the last BYE sent.

This event indicates that the session has been successfully terminated and that the session service does not need to be kept. However, the application should verify that no other dialog is active on this context through the `ISipUserAgentSvc` (see page 639) interface before releasing all its references to the context. For instance, it could be the case that the subscription to a transfer call result is not terminated yet.

See Also

`ISipSessionSvc` (see page 578), `ISipUserAgentSvc` (see page 639)

3.7.1.29 - ISipSessionSvc Class

Class Hierarchy



C++

```
class ISipSessionSvc : public IEComUnknown;
```

Description

The session service is used to establish a SIP dialog with a remote user agent by sending an INVITE. It is also used in the other direction to send progression status when a remote user agent tries to establish a dialog with the application by sending an INVITE.

This interface can also be used to send refreshes or modifications to the established dialog (or send the response to the refresh or modification initiated by the remote user agent).

Sending ACK Requests

In RFC 3261, the ACK request is used to stop the retransmission of final responses to an INVITE. An ACK sent to stop the transmission of a final error response (>=300) does not contain a payload while the ACK sent to stop the retransmissions of success responses (2xx) must contain a payload if the initial INVITE did not specify any offer. Thus, when the SIP Stack receives a final error response, it automatically sends an ACK to stop the retransmissions. When receiving a success response, the SIP Stack lets the application issue the ACK, since it may be sent with a content.

Terminating a Dialog

To tear down an established session, the application must call the `Bye` (see page 580) method.

This service generates events to the application through the `ISipSessionMgr` (see page 569) interface.

The ISipSessionSvc is an ECOM object

The `ISipSessionSvc` is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: `CLSID_CSipSessionSvc`

Interface Id: `IID_ISipSessionSvc`

A user can query the `ISipContext` (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the `ISipContext` (see page 23) through the same mean.

Updated Behaviour:

Some behaviours of the session service have changed from M5T SIP UA 3.x versions.

- Releasing the context or setting the manager to NULL does not terminate

all pending transactions in the session service. Instead, the application has to properly terminate all pending transactions before being done with the context. The application needs to call the `Bye` (see page 580) or `CancelRequest` method to properly terminate a session. When the session is terminated by the remote party, it needs to send a success final response because the SIP Stack does not automatically respond any final response to a `BYE` request.

- The responses to incoming requests are not sent through the session

service. They are sent through a new method `SendResponse` in the `ISipServerEventControl` (see page 75) interface.

- The `BYE` method does not send a `CANCEL` request. It can only be used

to send a BYE request. To send a CANCEL request, the `CancelRequest` method on the interface `ISipClientTransaction` (see page 21) has to be used.

Location

`SipUserAgent/ISipSessionSvc.h`

See Also

`ISipSessionMgr` (see page 569), `ISipContext` (see page 23)

Methods

Method	Description
• <code>Ack</code> (see page 579)	Sends an ACK request to the target server.
• <code>Bye</code> (see page 580)	Sends a BYE request to terminate this dialog.
• <code>HandleOriginalInviteResponseNewSession</code> (see page 580)	Handles the newly created context for original INVITE response.
• <code>Invite</code> (see page 581)	Sends an INVITE request to the target server.
• <code>SetManager</code> (see page 582)	Sets the manager of the session.

Legend

	Method
	abstract

3.7.1.29.1 - Methods

3.7.1.29.1.1 - `ISipSessionSvc::Ack` Method

Sends an ACK request to the target server.

C++

```
virtual mxt_result Ack(IN TOA CHeaderList* pExtraHeaders, IN TOA CSipMessageBody* pMessageBody) = 0;
```

Parameters

Parameters	Description
IN TOA CHeaderList* <code>pExtraHeaders</code>	The SIP headers to add in the packet. Ownership is TAKEN.
IN TOA CSipMessageBody* <code>pMessageBody</code>	The packet's body, aka payload. Ownership is TAKEN.

Returns

- `resFE_INVALID_STATE`: The manager of this session service is NULL, the session is not in a state where it is possible to send an ACK request, or it is impossible to create the request by using the User-Agent service. The manager of the service can be NULL after `Clear` is called on the context or when it was not set.
- `resFE_SIPCORE_PACKET_BLOCKED`: One of the service synchronously blocked the packet. No additional event will be reported.
- `resFE_FAIL`: Other failure that may have occurred.
- `resS_OK`: The ACK request is going to be sent.

Description

This method is used to acknowledge success final response (2xx response). If no success final response was received, the method returns an error. The payload may contain a session description.

See Also

`ISipSessionMgr` (see page 569), `CHeaderList` (see page 155), `CSipMessageBody` (see page 298), `ISipClientTransaction` (see page 21)

3.7.1.29.1.2 - ISipSessionSvc::Bye Method

Sends a BYE request to terminate this dialog.

C++

```
virtual mxt_result Bye(IN mxt_opaque opqTransaction, IN TOA CHeaderList* pExtraHeaders, IN TOA CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TOA CHeaderList* pExtraHeaders	The SIP headers to add in the packet. Ownership is TAKEN.
IN TOA CSipMessageBody* pMessageBody	The packet's body, aka payload. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

- **resFE_INVALID_STATE**: The manager of this session service is NULL, the session is not in a state where it is possible to send a BYE request or it is impossible to create the request using the User-Agent service. The manager of the service can be NULL after Clear is called on the context, or when it was not set.
- **resFE_SIPCORE_PACKET_BLOCKED**: One of the service synchronously blocked the packet. No additional event will be reported.
- **resFE_FAIL**: Other failure such as creating a transaction occurred.
- **resS_OK**: The BYE request is going to be sent.

Description

Use this method to terminate an established dialog by using a BYE request. It can also be used to terminate an early dialog established with an INVITE that was forked. Terminating a dialog can also be done with a CANCEL request by using ISipClientTransaction::CancelRequest (see page 22).

This method returns an error if the application tries to send a BYE while it is not in the proper state.

Upon receiving the event `EvTerminationSuccess`, it indicates that the session has been successfully terminated and that the session service does not need to be kept. However, the application should verify that no other dialog is active on this context through the ISipUserAgentSvc (see page 639) interface before releasing all its references to the context. For instance, it could be the case that the subscription to a transfer call result is not terminated yet.

Even if the API of ISipClientTransaction (see page 21) allows to cancel a pending BYE request, it should not be used to do so. RFC 3261 does not recommend to cancel a BYE request.

See Also

ISipSessionMgr (see page 569), CHeaderList (see page 155), CSipMessageBody (see page 298), ISipClientTransaction (see page 21), ISipUserAgentSvc (see page 639).

3.7.1.29.1.3 - ISipSessionSvc::HandleOriginalInviteResponseNewSession Method

Handles the newly created context for original INVITE response.

C++

```
virtual mxt_result HandleOriginalInviteResponseNewSession(IN mxt_opaque opqTransaction, IN ISipSessionSvc* pNewSession, IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN ISipSessionSvc* pNewSession	The newly created session. MUST NOT be NULL.
IN const CSipPacket& rPacket	The INVITE response packet.

Returns

- resS_OK: The initial INVITE response has been reported properly to the application.
- Other return codes: Failures.

Description

Notifies the session service that the application has created a new context to handle the given original INVITE response.

The newly created context MUST at least contain the User-Agent and session services. If one of those services is missing, this method fails.

3.7.1.29.1.4 - ISipSessionSvc::Invite Method

Sends an INVITE request to the target server.

C++

```
virtual mxt_result Invite(IN mxt_opaque opqTransaction, IN TOA CHeaderList* pExtraHeaders, IN TOA
CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TOA CHeaderList* pExtraHeaders	The SIP headers to add in the packet. Ownership is TAKEN.
IN TOA CSipMessageBody* pMessageBody	The packet's body, aka payload. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

- resFE_INVALID_STATE: The manager of this session service is NULL, there is an INVITE request underway, the User-Agent service is not properly configured (e.g.: contact list is empty) or it is impossible to create the request by using the User-Agent service. The manager of the service can be NULL after Clear is called on the context, or when it was not set.
- resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
- resFE_FAIL: Other failure such as creating a transaction occurred.
- resS_OK: The INVITE request is going to be sent.

Description

This method is used to establish or refresh a dialog with a remote user agent by sending an INVITE. Response to this INVITE is reported through the ISipSessionMgr (see page 569) interface. Usually the content contains a description of the session that the application is trying to establish (an SDP packet).

The reference to ISipClientTransaction (see page 21) allows users to further control the transaction. Note that it is possible to cancel

an INVITE request by using `ISipClientTransaction::CancelRequest` (see page 22).

See Also

`ISipSessionMgr` (see page 569), `CHeaderList` (see page 155), `CSipMessageBody` (see page 298), `ISipClientTransaction` (see page 21)

3.7.1.29.1.5 - `ISipSessionSvc::SetManager` Method

Sets the manager of the session.

C++

```
virtual mxt_result SetManager(IN ISipSessionMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipSessionMgr* pMgr	The manager. It cannot be NULL.

Returns

- `resFE_INVALID_ARGUMENT`: The manager cannot be NULL.
- `resS_OK`: The manager was set properly.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.30 - `ISipSessionTimerMgr` Class

Class Hierarchy

```
ISipSessionTimerMgr
```

C++

```
class ISipSessionTimerMgr;
```

Description

The session timer manager is the interface through which the session timer service talks to the application. The SIP stack notifies the application through this interface that a session needs to be refreshed or that it has expired.

Location

`SipUserAgent/ISipSessionTimerMgr.h`

See Also

`ISipSessionTimerSvc` (see page 584)

Methods

Method	Description
• A <code>EvSessionExpired</code> (see page 583)	Notifies the application that it should consider the session expired.
• A <code>EvSessionIntervalTooShortRecv</code> (see page 583)	Notifies the application that a 422 response has been received, meaning that the refresh rate requested by the application for this session is too short for the remote party.
• A <code>EvSessionIntervalTooShortSent</code> (see page 583)	Notifies the application that the refresh rate requested by the remote party for this session is too short.
• A <code>EvSessionMustRefresh</code> (see page 584)	Notifies the application that it must refresh the session.

Legend

	Method
	abstract

3.7.1.30.1 - Methods**3.7.1.30.1.1 - **ISipSessionTimerMgr::EvSessionExpired** Method**

Notifies the application that it should consider the session expired.

C++

```
virtual void EvSessionExpired(IN ISipSessionTimerSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipSessionTimerSvc* pSvc	The service managing the response.

Description

Notifies the application that the session should be considered expired. This method is called when two thirds of the time negotiated for the session has elapsed. When this method is called, it means that the user-agent responsible to do refreshes (either the local or the remote) failed to send the INVITE or UPDATE request in time.

The application should tear down the session (by releasing the context.)

See Also

ISipSessionTimerSvc (see page 584)

3.7.1.30.1.2 - **ISipSessionTimerMgr::EvSessionIntervalTooShortRecv Method**

Notifies the application that a 422 response has been received, meaning that the refresh rate requested by the application for this session is too short for the remote party.

C++

```
virtual void EvSessionIntervalTooShortRecv(IN ISipSessionTimerSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionTimerSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
IN const CSipPacket& rRequest	The response.

Description

Notifies the application that a 422 response has been received, meaning that the refresh rate requested by the application for this session is too short for the remote party. The application must therefore issue another INVITE or UPDATE request with a higher Session-Expires value.

See Also

ISipSessionTimerSvc (see page 584), ISipSessionTimerSvc::SetSessionExpiresSec (see page 588)

3.7.1.30.1.3 - **ISipSessionTimerMgr::EvSessionIntervalTooShortSent Method**

Notifies the application that the refresh rate requested by the remote party for this session is too short.

C++

```
virtual void EvSessionIntervalTooShortSent(IN ISipSessionTimerSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest) = 0;
```

Parameters

Parameters	Description
IN ISipSessionTimerSvc* pSvc	The service managing the response.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rRequest	The response.

Description

Notifies the application that the session interval requested by the remote party was shorter than the configured minimum and that, as a result, the service has issued a 422 response code through the request context associated with the transaction. Therefore, there's no need for the application to send a response, it has already been sent by the service using ISipServerEventCtrl::SendResponse.

See Also

ISipSessionTimerSvc (see page 584), ISipServerEventControl::SendResponse (see page 76)

3.7.1.30.1.4 - ISipSessionTimerMgr::EvSessionMustRefresh Method

Notifies the application that it must refresh the session.

C++

```
virtual void EvSessionMustRefresh(IN ISipSessionTimerSvc* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipSessionTimerSvc* pSvc	The service managing the response.

Description

Notifies the application that the session needs to be refreshed. If session-timer negotiation has determined that the local user-agent was responsible for doing the refreshes, this method is called when half the time negotiated for the session duration has elapsed.

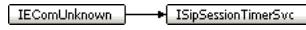
In order to keep the session alive, when this method is called, the application should send an INVITE or an UPDATE request to the remote user agent.

See Also

ISipSessionTimerSvc (see page 584)

3.7.1.31 - ISipSessionTimerSvc Class

Class Hierarchy



C++

```
class ISipSessionTimerSvc : public IEComUnknown;
```

Description

The session-timer service is used to give a keep alive mechanism to the session described in the parent context. It negotiates a session duration by inspecting and modifying INVITE requests and their responses (sent by other services). It then warns the application (through the ISipSessionTimerMgr (see page 582) interface) when it is time to refresh the session in order to keep it alive or when it is time to tear down the session because it has expired.

This implementation is compliant with RFC 4028.

This service manager generates events to the application through the ISipSessionTimerMgr (see page 582) interface.

The ISipSessionTimerSvc is an ECOM object

The ISipSessionTimerSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipSessionTimerSvc**Interface Id: IID_ISipSessionTimerSvc**

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipSessionTimerSvc.h

Methods

Method	Description
◆ A GetMinSESec (see page 585)	Gets the current Min-SE value.
◆ A GetSessionExpiresSec (see page 585)	Gets the current Session-Expires value.
◆ A Reset (see page 586)	Resets the service to its initial values.
◆ A ResetSessionTimer (see page 586)	Resets the session timer to avoid generating EvSessionExpired at 2/3 of the negotiated session-expire.
◆ A SetExpirationThresholds (see page 586)	Sets thresholds before which the application is notified of upcoming or effective session expiration.
◆ A SetManager (see page 587)	Configures the event manager associated with this service.
◆ A SetMinSESec (see page 587)	Configures the Min-SE required value.
◆ A SetMissingSessionExpiresInAnswerBehaviour (see page 588)	
◆ A SetRefresher (see page 588)	Configures the required refresher.
◆ A SetSessionExpiresSec (see page 588)	Configures the Session-Expires required value.
◆ A SetSessionTimerDemanded (see page 589)	Sets whether or not Session-Expires and Min-SE headers must be added to requests.

Legend

◆	Method
A	abstract

Enumerations

Enumeration	Description
EMissingSessionExpireBehaviour (see page 589)	Possible behaviour to set with SetMissingSessionExpiresInAnswerBehaviour (see page 588).
ERefresher (see page 590)	

3.7.1.31.1 - Methods**3.7.1.31.1.1 - ISipSessionTimerSvc::GetMinSESec Method**

Gets the current Min-SE value.

C++

```
virtual unsigned int GetMinSESec() = 0;
```

Returns

The current minimum amount of time (in seconds) after which the session should be considered expired if not refreshed.

Description

This method can be used to get the Min-SE current value.

3.7.1.31.1.2 - ISipSessionTimerSvc::GetSessionExpiresSec Method

Gets the current Session-Expires value.

C++

```
virtual unsigned int GetSessionExpiresSec() = 0;
```

Returns

The current maximum amount of time (in seconds) after which the session should be considered expired if not refreshed.

Description

This method can be used to get the current Session-Expires value.

3.7.1.31.1.3 - ISipSessionTimerSvc::Reset Method

Resets the service to its initial values.

C++

```
virtual void Reset() = 0;
```

Description

Resets the service to its initial values. Useful when the application wants to renegotiate session expiration when refreshing the session.

3.7.1.31.1.4 - ISipSessionTimerSvc::ResetSessionTimer Method

Resets the session timer to avoid generating EvSessionExpired at 2/3 of the negotiated session-expire.

C++

```
virtual mxt_result ResetSessionTimer() = 0;
```

Returns

- resFE_UNEXPECTED: The refresher is not set or is remote.
- resFE_INVALID_STATE: The function is called before the session-expire parameters are negotiated or after the context has been cleared.
- resS_OK: The function has restarted the session timer successfully.

Description

Re-starts the session timers using the values negotiated when the last 2xx response was received.

This method should be called only when a final negative response is being received to an INVITE or UPDATE sent to refresh the session timers. In such case, the event EvFailure is triggered on the ISipSessionMgr (see page 569).

This feature allows to handle specific scenarios, such as receiving a 488 during a session timer refresh. Calling this method will avoid generating the event EvSessionExpired at 2/3 of the negotiated session-expire values.

This method only re-starts the timers when at least one 2xx to an INVITE has been received. It fails if called on the non-refresher side or after that the context has been cleared.

3.7.1.31.1.5 - ISipSessionTimerSvc::SetExpirationThresholds Method Updated behavior in 4.1.4

Sets thresholds before which the application is notified of upcoming or effective session expiration.

C++

```
virtual mxt_result SetExpirationThresholds(IN unsigned int uRefreshThresholdSec, IN unsigned int uExpiredThresholdSec) = 0;
```

Parameters

Parameters	Description
IN unsigned int uRefreshThresholdSec	The time before expiration at which EvSessionMustRefresh is called (in seconds). 0 means to keep the default.
IN unsigned int uExpiredThresholdSec	The time before expiration at which EvSessionExpired is called (in seconds). 0 means to keep the default.

Returns

resFE_INVALID_ARGUMENT: uExpiredThresholdSec is greater than or equal to uRefreshThresholdSec. resFE_INVALID_STATE: If

the manager is not set.

Description

Sets the amount of time that should remain to a session before `EvSessionMustRefresh` and `EvSessionExpired` are called for that session.

Note this service will notify its manager of required refreshes only if the refresher is local (via `EvSessionMustRefresh`). Yet, it will notify it of expirations (via `EvSessionExpired`) regardless of the refresher status.

Note that in RFC 4028 it is RECOMMENDED, regarding refreshes and expiration:

- Refreshes (Section 7.2): should be sent after one half the session interval has elapsed, by the refresher.
- Expiration (Section 10): `BYE` should be sent one third of the session interval or 32 seconds, whichever is lower, before expiration, by the side not performing refreshes.

Thus, by default `EvSessionMustRefresh` will be called after half the session interval has elapsed but only if the refresher is local. `EvSessionExpired` will be called after two thirds of the session interval has elapsed if it is 96 seconds or less. If it is more than 96 seconds, `EvSessionExpired` will be called 32 seconds before expiration. This is in line with RFC 4028 and should be adequate for most applications.

However, this method is provided to modify this default behaviour. The parameters are used to tell the service how long before expiration it wants to be notified.

Changing expiration thresholds does not affect the current session, if any is already established. It affects the next session established.

Updated behavior: Previously, `EvSessionExpired` was fired by default after two thirds of the session interval, regardless of the 32 seconds rule. For instance, a session with an interval of 180 seconds would receive `EvSessionExpired` after 120 seconds and not 148 seconds. This has been fixed and the behaviour is now aligned with RFC 4028.

See Also

`ISipSessionTimerMgr::EvSessionMustRefresh` (see page 584), `ISipSessionTimerMgr::EvSessionExpired` (see page 583)

3.7.1.31.1.6 - **ISipSessionTimerSvc::SetManager** Method

Configures the event manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipSessionTimerMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipSessionTimerMgr* pMgr	The event manager. It must not be NULL.

Description

Configures the event manager that will receive the events generated by this service.

Note that an event manager MUST be associated with this service before it is used.

3.7.1.31.1.7 - **ISipSessionTimerSvc::SetMinSESec** Method

Configures the Min-SE required value.

C++

```
virtual mxt_result SetMinSESec(IN unsigned int uMinSESec = uDEFAULT_MIN_SE_SEC) = 0;
```

Parameters

Parameters	Description
IN unsigned int uMinSESec = uDEFAULT_MIN_SE_SEC	The minimum required amount of time (in seconds) after which the session should be considered expired if not refreshed. Default to 90 seconds. It cannot be less than 90 seconds.

Returns

resSW_WARNING: uMinSESec is higher than the required Session-Expires value or lower than 90 seconds.

resS_OK: Otherwise.

Description

This method can be used to set the Min-SE required value. If the value passed in parameter is higher than the required Session-Expires value (set with SetSessionExpiresSec (see page 588)), the Session-Expires value is set to the new minimum. If the value is lower than 90 seconds, it is set to 90 seconds.

Notes

This method should be called before SetSessionExpiresSec (see page 588)() since setting the minimum value has an impact on the validity of the Session-Expires value.

See Also

SetSessionExpiresSec (see page 588)

3.7.1.31.1.8 - ISipSessionTimerSvc::SetMissingSessionExpiresInAnswerBehaviour Method

```
virtual void SetMissingSessionExpiresInAnswerBehaviour(IN EMissingSessionExpireBehaviour eBehaviour) = 0;
```

Parameters

Parameters	Description
IN EMissingSessionExpireBehaviour eBehaviour	<ul style="list-style-type: none"> eRefreshSession: 200 OK without Session-Expires will always refresh the session eStopSessionTimer: 200 OK without Session-Expires stops the timer. Hence no more EvSessionMustRefresh and EvSessionExpired are reported.

Description

If eRefreshSession is passed, the service will reset the timer (hence reports EvMustRefresh when the timer fires) on the reception of all 200 OK. If eStopSessionTimer, the timer will be reset only if the 200 OK contains the header Session-Expires. By default, the value is eRefreshSession.

3.7.1.31.1.9 - ISipSessionTimerSvc::SetRefresher Method

Configures the required refresher.

C++

```
virtual void SetRefresher(ERefresher eRefresher = eNONE) = 0;
```

Parameters

Parameters	Description
ERefresher eRefresher = eNONE	The party the application wants to act as the refresher, if any.

Description

This method can be used to set the required refresher.

3.7.1.31.1.10 - ISipSessionTimerSvc::SetSessionExpiresSec Method

Configures the Session-Expires required value.

C++

```
virtual mxt_result SetSessionExpiresSec(IN unsigned int uSessionExpiresSec = uDEFAULT_SESSION_EXPIRES_SEC) = 0;
```

Parameters

Parameters	Description
IN unsigned int uSessionExpiresSec = uDEFAULT_SESSION_EXPIRES_SEC	The maximum required amount of time (in seconds) after which the session should be considered expired if not refreshed. Default to 30 minutes.

Returns

resSW_WARNING: uSessionExpiresSec is lower than the required Min-SE value.

resS_OK: Otherwise.

Description

This method can be used to set the Session-Expires required value. If the value passed in parameter is lower than the required Min-SE value (set with SetMinSESec (see page 587)), it is set to the Min-SE value. Otherwise, it is set to the value passed in parameter.

Notes

If SetMinSESec (see page 587)() is called after this method with a minimal value lower than uSessionExpiresSec, the Session-Expires value will be changed to the new minimum. For that reason, it is recommended to call this method after SetMinSESec().

See Also

SetMinSESec (see page 587)

3.7.1.31.1.11 - ISipSessionTimerSvc::SetSessionTimerDemanded Method

Sets whether or not Session-Expires and Min-SE headers must be added to requests.

C++

```
virtual void SetSessionTimerDemanded(IN bool bSEDemanded = false, IN bool bMinSEDemanded = false) = 0;
```

Parameters

Parameters	Description
IN bool bSEDemanded = false	True if the application wants to add a Session-Expires header to requests, false if it just wants to signal that the session timer capability is supported.
bMinSE	True if the application wants to add a Min-SE header to requests, false otherwise.

Description

This method can be used to set whether or not Session-Expires and Min-SE headers must be added to requests.

3.7.1.31.2 - Enumerations

3.7.1.31.2.1 - ISipSessionTimerSvc::EMissingSessionExpireBehaviour Enumeration

Possible behaviour to set with SetMissingSessionExpiresInAnswerBehaviour (see page 588).

C++

```
enum EMissingSessionExpireBehaviour {
    eRefreshSession,
    eStopSessionTimer
};
```

Description

Indicates the behaviour when receiving a 200 OK without the Session-Expires header.

Members

Members	Description
eRefreshSession	200 OK without Session-Expires will always refresh the session.
eStopSessionTimer	200 OK without Session-Expires stops the timer. Hence no more EvSessionMustRefresh and EvSessionExpired are reported.

3.7.1.31.2.2 - ISipSessionTimerSvc::ERefresher Enumeration

```
enum ERefresher {
    eNONE,
    eLOCAL,
    eREMOTE
};
```

Description

Enum to determine who must refresh the session.

Members

Members	Description
eNONE	No one will refresh.
eLOCAL	The local entity will refresh the session.
eREMOTE	The remote end will refresh the session.

3.7.1.32 - ISipSubscriberMgr Class New in 4.1.4

Class Hierarchy

```
[ ISipSubscriberMgr ]
```

C++

```
class ISipSubscriberMgr;
```

Description

The interface through which the ISipSubscriberSvc (see page 595) reports events to the application.

Warning

This class replaces the old class of the same name, which has been deprecated and renamed ISipSubscriberBaseMgr for clarity.

Location

SipUserAgent/ISipSubscriberMgr.h

See Also

ISipSubscriberSvc (see page 595)

Methods

Method	Description
• A EvExpired (see page 591)	The subscription is expired.
• A EvExpiring (see page 591)	The subscription is about to expire.
• A EvFailure (see page 591)	A failure (>= 3xx) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).
• A EvIntervalTooSmall (see page 592)	An "Interval Too Small" (423) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).
• A EvInvalidNotify (see page 592)	An invalid NOTIFY has been received.
• A EvNotified (see page 593)	A NOTIFY request has been received for a valid active subscription.
• A EvProgress (see page 594)	A provisional response (1xx) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).
• A EvSuccess (see page 594)	A success response (2xx) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).
• A EvTerminated (see page 595)	A final NOTIFY has been received for a valid active subscription.

Legend

• A	Method
A	abstract

3.7.1.32.1 - Methods

3.7.1.32.1.1 - **ISipSubscriberMgr::EvExpired** Method

The subscription is expired.

C++

```
virtual void EvExpired(IN ISipSubscriberSvc* pSvc, IN const CString& rstrEvent, IN const CString& rstrId) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface that has an expired subscription.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id of the expired subscription.

Description

Notifies the application that a subscription is expired. There is no action to take by the application as far as the ISipSubscriberSvc (see page 595) is concerned.

Note that this event can happen for a subscription created with CreateSubscription, Subscribe and will happen if Fetch does not receive a NOTIFY with "subscription-state" "terminated" within 32 seconds.

See Also

EvExpiring (see page 591)

3.7.1.32.1.2 - **ISipSubscriberMgr::EvExpiring** Method

The subscription is about to expire.

C++

```
virtual void EvExpiring(IN ISipSubscriberSvc* pSvc, IN const CString& rstrEvent, IN const CString& rstrId) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface that has a subscription about to expire.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id of the subscription about to expire.

Description

Notifies the application that a subscription will expire in the amount of time configured with the ISipSubscriberSvc::SetExpiringThreshold (see page 600) method. This event does not occur when a subscription is set for a time smaller than or equal to the threshold.

The application should then either call Refresh to keep the subscription active or Terminate to terminate it.

See Also

EvExpired (see page 591)

3.7.1.32.1.3 - **ISipSubscriberMgr::EvFailure** Method

A failure (>= 3xx) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).

C++

```
virtual void EvFailure(IN ISipSubscriberSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.

IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id parameter of the Event header of the SUBSCRIBE request.
IN const CSipPacket& rResponse	The failure response received.

Description

Notifies the application that a failure response (>=300) has been received for a SUBSCRIBE request that it sent. Note that this SUBSCRIBE request could result from a call to Subscribe, Fetch, Refresh, or Terminate.

See Also

ISipClientEventControl (see page 19), ISipSubscriberSvc (see page 595)

3.7.1.32.1.4 - ISipSubscriberMgr::EvIntervalTooSmall Method

An "Interval Too Small" (423) has been received for a SUBSCRIBE request sent by the ISipSubscriberSvc (see page 595).

C++

```
virtual void EvIntervalTooSmall(IN ISipSubscriberSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN
unsigned int uMinExpirationSec, IN const CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket&
rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. Note however that RelIssueRequest would not correct the problem because the SUBSCRIBE request would be retried with the same expiration time.
IN unsigned int uMinExpirationSec	The value of the Min-Expires header found in the 423 response.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id parameter of the Event header of the SUBSCRIBE request.
IN const CSipPacket& rResponse	The success response received.

Description

Informs the application that a 423 Interval Too Small response has been received for a SUBSCRIBE request that it sent. This SUBSCRIBE was sent through Subscribe or Refresh. The application should call the same method but with an expiration time greater than or equal to uMinExpirationSec.

See Also

ISipClientEventControl (see page 19), ISipSubscriberSvc (see page 595)

3.7.1.32.1.5 - ISipSubscriberMgr::EvInvalidNotify Method

An invalid NOTIFY has been received.

C++

```
virtual void EvInvalidNotify(IN ISipSubscriberSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket&
rNotify, IN mxt_result reason) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface that handled the notify.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rNotify	The NOTIFY request received.

<pre>IN mxt_result reason</pre>	<ul style="list-style-type: none"> • resFE_UNKNOWN_EVENT: The Event header was missing or for an unknown event type. A "489 Bad Event" response has already been sent. • resFE_UNKNOWN_SUBSCRIPTION: The NOTIFY contains an id unknown to the service. A "481 Subscription does not exist" response has already been sent. • resFE_EXPIRED_SUBSCRIPTION: The NOTIFY has been received after another NOTIFY with "subscription-state" "terminated" or after the expiration time. A "481 Subscription does not exist" response has already been sent. • resFE_MISSING_HEADER: The received NOTIFY did not contain the mandatory "Subscription-State" header.
---------------------------------	--

Description

Notifies the application that an invalid NOTIFY has been received. Usually, the application has no action to take other than reporting this invalid packet if wanted. The ISipSubscriberSvc (see page 595) that reported the event already took care of sending the appropriate response.

One action that can be taken is to destroy the context if it was created only to handle this NOTIFY as it will not create a dialog.

See Also

EvNotified (see page 593), EvTerminated (see page 595)

3.7.1.32.1.6 - ISipSubscriberMgr::EvNotified Method

A NOTIFY request has been received for a valid active subscription.

C++

```
virtual void EvNotified(IN ISipSubscriberSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rNotify) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the subscription has been created.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id of the subscription. It can be an empty string, which means that there is no id.
IN const CSipPacket& rNotify	The NOTIFY request received.

Description

Notifies the application that a valid NOTIFY request has been received. The ISipSubscriberSvc (see page 595) has already validated that the NOTIFY is for an active subscription and that it is valid. However, the ISipSubscriberSvc (see page 595) does not automatically accept this NOTIFY request with a 200 OK response because it does not verify the content and the content could be unacceptable to the application.

Note that this NOTIFY request is guaranteed to contain a "subscription-state" header and its value is guaranteed not to be "terminated" either.

When a NOTIFY with a "subscription-state" header with value "terminated" is received, it is presented to the application through the EvTerminated (see page 595) method.

The first NOTIFY sent after a SUBSCRIBE has to contain a message- summary. The draft does not specify that the following NOTIFYs have to contain a message-summary. If a message-summary is wrong or not present when it should be, a 400 response is sent back

automatically. If the content-type is wrong, a 415 response is sent back automatically.

See Also

[EvTerminated](#) (see page 595)

3.7.1.32.1.7 - **ISipSubscriberMgr::EvProgress** Method

A provisional response (1xx) has been received for a SUBSCRIBE request sent by the [ISipSubscriberSvc](#) (see page 595).

C++

```
virtual void EvProgress(IN ISipSubscriberSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id parameter of the Event header of the SUBSCRIBE request.
IN const CSipPacket& rResponse	The provisional response received.

Description

Notifies the application that a provisional response has been received for a SUBSCRIBE request that it sent. Note that this SUBSCRIBE request could result from a call to [Subscribe](#), [Fetch](#), [Refresh](#), or [Terminate](#).

See Also

[ISipClientEventControl](#) (see page 19), [ISipSubscriberSvc](#) (see page 595)

3.7.1.32.1.8 - **ISipSubscriberMgr::EvSuccess** Method

A success response (2xx) has been received for a SUBSCRIBE request sent by the [ISipSubscriberSvc](#) (see page 595).

C++

```
virtual void EvSuccess(IN ISipSubscriberSvc* pSvc, IN ISipClientEventControl* pClientEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the SUBSCRIBE request has been sent. It cannot be NULL.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelissueRequest on it. Note however that RelissueRequest is usually called to retry failed requests instead of successful ones.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id parameter of the Event header of the SUBSCRIBE request.
IN const CSipPacket& rResponse	The success response received.

Description

Informs the application that a success response has been received for a SUBSCRIBE request that it sent. If that SUBSCRIBE request was sent through [Subscribe](#) or [Refresh](#), the application should consider that the subscription is active and be ready to receive [EvNotified](#) (see page 593) events. If the SUBSCRIBE request was sent through a [Fetch](#) or a [Terminate](#), the application should expect to receive the event [EvTerminated](#) (see page 595) or [EvExpired](#) (see page 591). Note that the [EvTerminated](#) (see page 595) or [EvExpired](#) (see page 591) can still occur unrequested if the notifier decides to terminate the subscription or if the subscription expires.

See Also

[ISipClientEventControl](#) (see page 19), [ISipSubscriberSvc](#) (see page 595)

3.7.1.32.1.9 - ISipSubscriberMgr::EvTerminated Method

A final NOTIFY has been received for a valid active subscription.

C++

```
virtual void EvTerminated(IN ISipSubscriberSvc* pSvc, IN ISipServerEventControl* pServerEventControl, IN const
CString& rstrEvent, IN const CString& rstrId, IN const CSipPacket& rNotify) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberSvc* pSvc	The interface on which the subscription has been created.
IN ISipServerEventControl* pServerEventControl	The server event control interface for this transaction. It cannot be NULL. The application must call SendResponse with a final response on it.
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN const CString& rstrId	The id of the subscription. It can be an empty string, which means that there is no id.
IN const CSipPacket& rNotify	The NOTIFY request received.

Description

Informs the application that a valid NOTIFY request has been received. The ISipSubscriberSvc (see page 595) has already validated that the NOTIFY is for an active subscription and that it is valid. However, the ISipSubscriberSvc (see page 595) does not automatically accept this NOTIFY request with a 200 OK response because it does not verify the content and the content could be unacceptable to the application.

If accepted, this NOTIFY terminates the subscription. It is guaranteed to contain a "subscription-state" header with the value "terminated".

The first NOTIFY sent after a SUBSCRIBE has to contain a message- summary. The draft does not specify that the following NOTIFYS have to contain a message-summary. If a message-summary is wrong or not present when it should be, a 400 response is sent back automatically. If the content-type is wrong, a 415 response is sent back automatically.

See Also

EvNotified (see page 593)

3.7.1.33 - ISipSubscriberSvc Class New in 4.1.4 | Updated behavior in 4.1.4

Class Hierarchy



C++

```
class ISipSubscriberSvc : public IEComUnknown;
```

Description

This interface is used to act as a subscriber as defined in RFC 3265. It lets the application send SUBSCRIBE and receive NOTIFY for multiple Events. By default, the subscriber service does not handle any event types and will reject all subscription requests. Supported events are added through the AddEvent (see page 597) method. Once event types are thus added they are correctly handled.

This service reports events received to the application through the ISipSubscriberMgr (see page 590) interface.

The ISipSubscriberSvc is an ECOM object

The ISipSubscriberSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipSubscriberSvc

Interface Id: IID_ISipSubscriberSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Updated behavior: Previously, each event type had its own specific subscriber and notifier services, with its own interface and

implementation classes. Thus, for a SIP context to support different event types, many subscriber and/or notifier services had to be individually created and attached to the associated `ISipContext` (see page 23). Moreover, new event types required the creation of new child classes deriving from base subscriber or notifier classes, along with supporting infrastructure. This scheme was cumbersome and made supporting new event types unnecessarily complex.

It follows from the foregoing that:

- Supporting many different event types in the same `ISipContext` (see page 23) now requires creating a single subscription or notification service, upon which `AddEvent` (see page 597)() is called many times (once for each event type to support). The service is then added to the context. This previously required many different services to be created and attached.
- Existing user-defined subscription or notification services should be removed from the code base and replaced with the new subscriber or notifier service. Of course, appropriate calls to `AddEvent` (see page 597)() should also be made. For instance, an "ISipSubscriberProprietaryEventSvc" subscription service should be replaced with a plain `ISipSubscriberSvc` upon which `AddEvent` (see page 597)("ProprietaryEvent", ...) is called.
- Adding new user-defined subscription event types is now trivial.

Warning

Old subscriber and notifier services are deprecated and are now removed. By old services we refer to interfaces that were previously named `ISipNotifier[EventName]Svc` and `ISipSubscriber[EventName]Svc`.

Location

`SipUserAgent/ISipSubscriberSvc.h`

See Also

`ISipSubscriberMgr` (see page 590)

Methods

Method	Description
◆ A AbortSubscription (see page 596)	Terminates a subscription without sending a SUBSCRIBE request.
◆ A AddEvent (see page 597)	Adds an event type to be supported by this service.
◆ A CreateSubscription (see page 598)	Creates a subscription without sending a SUBSCRIBE request.
◆ A Fetch (see page 599)	Sends a SUBSCRIBE request only to obtain a single NOTIFY.
◆ A Refresh (see page 599)	Extends an active subscription.
◆ A SetExpiringThreshold (see page 600)	Sets the time to warn the application before a subscription expires.
◆ A SetManager (see page 601)	Configures the manager of this service.
◆ A Subscribe (see page 601)	Sends a SUBSCRIBE request to establish a subscription.
◆ A Terminate (see page 602)	Terminates an active subscription.

Legend

◆	Method
◆ A	abstract

Enumerations

Enumeration	Description
EType (see page 603)	

3.7.1.33.1 - Methods

3.7.1.33.1.1 - `ISipSubscriberSvc::AbortSubscription` Method

Terminates a subscription without sending a SUBSCRIBE request.

C++

```
virtual mxt_result AbortSubscription(IN const CString& rstrEvent, IN const CString& rstrId) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
strId	The "id" parameter that should be found in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter must not be in the NOTIFY to match that subscription.

Returns

- resFE_INVALID_ARGUMENT: There is no subscription with this id in this object or the event is unknown.
- resFE_INVALID_STATE: The subscription for this id cannot be aborted because it is already terminated.
- resFE_INVALID_STATE: The manager of the service was not set or Clear has been called on the context.
- resS_OK: The subscription for this id is correctly aborted.

Description

Terminates a subscription without sending a SUBSCRIBE request with the Expires header field set to 0. This method should be called only when an explicit subscription was created with the CreateSubscription (see page 598) method but the non-SUBSCRIBE request that was supposed to create the subscription received a final negative response.

When the method is called for an active implicit subscription (created through the CreateSubscription (see page 598) method with the eType parameter set to eIMPLICIT), it has the exact same effect as calling Terminate (see page 602) for that subscription.

When the method is called for an active subscription created through the Subscribe (see page 601) method, it has the effect of being non conformant with RFC 3265 since it terminates the subscription without sending a SUBSCRIBE request with the Expires header set to 0.

In every other case, the method returns an error.

See Also

CreateSubscription (see page 598), Subscribe (see page 601), Terminate (see page 602), AddEvent (see page 597)

3.7.1.33.1.2 - ISipSubscriberSvc::AddEvent Method

Adds an event type to be supported by this service.

C++

```
virtual mxt_result AddEvent(IN const CString& rstrEvent, IN unsigned int uDefaultExpiration) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	Event type to support, e.g., "message-summary".
IN unsigned int uDefaultExpiration	Default expiration for this event type, in seconds.

Returns

resS_OK: Success.

Description

Adds an event type to be supported by this service.

3.7.1.33.1.3 - ISipSubscriberSvc::CreateSubscription Method

Creates a subscription without sending a SUBSCRIBE request.

C++

```
virtual mxt_result CreateSubscription(IN const CString& rstrEvent, IN const CString& rstrId, IN unsigned int
uExpirationSec, IN EType eType, IN TO CGenParamList* pCustomParameters) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	Event type for which to create the subscription, e.g., "message-summary".
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration for that event type is used.
IN EType eType	Can be eEXPLICIT, which means that the subscription has been established by SIP signaling but by other means than a SUBSCRIBE request. When the subscription is not established via SIP, the value of this parameter should be eIMPLICIT.
IN TO CGenParamList* pCustomParameters	A pointer to the parameters added to the Event header. The pointer may be NULL. Ownership is TAKEN. This list should not contain the parameter "id" since it will be added from the strId parameter. If the "id" parameter is present, its value is replaced with strId.
strId	The "id" parameter that should be found in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter must not be in the NOTIFY to match that subscription.

Returns

- resFE_INVALID_ARGUMENT: The id is already used by another subscription in this object or the event is unknown.
- resFE_INVALID_STATE: The manager of the service was not set or Clear has been called on the context.
- resS_OK: The subscription is successfully created and ready to receive NOTIFY requests.

Description

Creates a subscription without sending a SUBSCRIBE request. Set the eType parameter to eEXPLICIT when the subscription is created by some other SIP means (eg. REFER request). Set the eType parameter to eIMPLICIT when the subscription is not created via SIP, such as a common agreement. The value of this parameter dictates what is to be done when the subscription is terminated or refreshed. When the subscription is explicit, calling Terminate (see page 602) or Refresh (see page 599) triggers a SUBSCRIBE request to be sent. When the subscription is implicit, calling those methods only has impact on the internal state and does not trigger a SUBSCRIBE request to be sent.

When establishing an explicit subscription through a non-SUBSCRIBE request, the application must not wait for the response to that request to call CreateSubscription. This is because the first NOTIFY could arrive before the final response to the request (because of forking and record routing). However, should the method that was supposed to create a subscription fails, the application should call AbortSubscription (see page 596) for this subscription as it will never receive a NOTIFY.

This method should also be used when the application wants to handle supplemental branches of a forked SUBSCRIBE. In this case, the NOTIFY can arrive with a different From tag than the To tag in the response to the SUBSCRIBE. In order for the subscriber service on the new context to accept that NOTIFY, the application must call CreateSubscription with the proper parameters.

Otherwise, the subscription created by this method works exactly the same way as the one created by the Subscribe (see page 601) method.

See Also

AbortSubscription (see page 596), Subscribe (see page 601), AddEvent (see page 597)

3.7.1.33.1.4 - ISipSubscriberSvc::Fetch Method

Sends a SUBSCRIBE request only to obtain a single NOTIFY.

C++

```
virtual mxt_result Fetch(IN const CString& rstrEvent, IN const CString& rstrId, IN mxt_opaque opqTransaction, IN
TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, IN TO CGenParamList* pCustomParameters, OUT
ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership is TAKEN.
IN TO CGenParamList* pCustomParameters	A pointer to the parameters added to the Event header. The pointer may be NULL. Ownership is TAKEN. This list should not contain the parameter "id" since it will be added from the strId parameter. If the "id" parameter is present, its value is replaced with strId.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
strId	The "id" parameter to be put in the Event header field of the SUBSCRIBE request sent. It should also be in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter is not in the SUBSCRIBE sent and it must not be in the NOTIFY to match that subscription.

Returns

- resFE_INVALID_ARGUMENT: The id is already used by another subscription in this object or the event is unknown.
- resFE_INVALID_STATE: The manager of the service was not set, Clear has been called on the context or the attached user agent service is not properly configured (e.g.: contact list is empty).
- resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
- resFE_FAIL: The SUBSCRIBE request could not be sent.
- resS_OK: The SUBSCRIBE request is successfully sent.

Description

Sends a SUBSCRIBE request but with the Expires header set to 0. If the request is successful, it makes the notifier send only a single NOTIFY (that is reported through the ISipSubscriberMgr (see page 590)). This method does not create a subscription. It cannot be used to terminate an active subscription neither.

See Also

AddEvent (see page 597)

3.7.1.33.1.5 - ISipSubscriberSvc::Refresh Method

Extends an active subscription.

C++

```
virtual mxt_result Refresh(IN const CString& rstrEvent, IN const CString& rstrId, IN unsigned int
uExpirationSec, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody*
pMessageBody, OUT ISipClientTransaction*& rpTransaction, IN TOA CGenParamList* pCustomParameters = NULL) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration for that event type is used.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
IN TOA CGenParamList* pCustomParameters = NULL	A pointer to the parameters added to the Event header. The pointer may be NULL, in which case, the previous set of parameters is used. Otherwise, if this parameter is not NULL, the new set of parameters is used. In the special case when pCustomParameters is not NULL but no parameters are specified in pCustomParameters, all the previous parameters are removed and no parameter is used. Ownership is TAKEN. This list should not contain the parameter "id" since it will be added from the strId parameter. If the "id" parameter is present, its value is replaced with strId.
strId	The "id" parameter that is put in the Event header field of the SUBSCRIBE request sent. It should also be in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter is not in the SUBSCRIBE sent and it must not be in the NOTIFY to match that subscription.

Returns

- **resFE_INVALID_ARGUMENT:** The id does not correspond to any subscription in this object or the event is unknown.
- **resFE_INVALID_STATE:** The manager of the service was not set or Clear has been called on the context.
- **resFE_SIPCORE_PACKET_BLOCKED:** One of the service synchronously blocked the packet. No additional event will be reported.
- **resFE_FAIL:** The SUBSCRIBE request could not be sent.
- **resS_OK:** The SUBSCRIBE request is successfully sent.

Description

This method acts exactly as [Subscribe](#) (see page 601) except that it can only be called on an active subscription. If successful, it extends the amount of time for which the subscription remains active.

See Also

[Subscribe](#) (see page 601), [AddEvent](#) (see page 597)

3.7.1.33.1.6 - ISipSubscriberSvc::SetExpiringThreshold Method Updated behavior in 4.1.4

Sets the time to warn the application before a subscription expires.

C++

```
virtual mxt_result SetExpiringThreshold(IN const CString& rstrEvent, IN unsigned int uThresholdSec) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN unsigned int uThresholdSec	The time, in seconds, before expiration time that the EvExpiring is fired.

Returns

resS_OK: Success. resFE_INVALID_ARGUMENT: The event type is unknown to this service.

Description

Sets the amount of time that should remain to a subscription before EvExpiring is called for that subscription.

This method affects only future subscriptions. This means that if there is an active subscription while this method is called, the EvExpiring event is called according to the previous threshold value.

If the expires value is smaller than or equal to the threshold, the EvExpiring event is never called, EvExpired is instead called directly.

See Also

Refresh (see page 599), Terminate (see page 602), AddEvent (see page 597)

3.7.1.33.1.7 - ISipSubscriberSvc::SetManager Method

Configures the manager of this service.

C++

```
virtual mxt_result SetManager(IN ISipSubscriberMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipSubscriberMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events from this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.33.1.8 - ISipSubscriberSvc::Subscribe Method

Sends a SUBSCRIBE request to establish a subscription.

C++

```
virtual mxt_result Subscribe(IN const CString& rstrEvent, IN const CString& rstrId, IN unsigned int uExpirationSec, IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, IN TO CGenParamList* pCustomParameters, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	Event type for which to create the subscription, e.g., "message-summary".
IN unsigned int uExpirationSec	The maximum time, in seconds, for which the subscription is considered active without refreshing. When 0, the default expiration for that event type is used.
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.

IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership is TAKEN.
IN TO CGenParamList* pCustomParameters	A pointer to the parameters to be added to the Event header. The pointer may be NULL. Ownership is TAKEN. This list should not contain the parameter "id" since it will be added from the strId parameter. If the "id" parameter is present, its value is replaced with strId.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
strId	The "id" parameter put in the Event header field of the SUBSCRIBE request sent. It should also be in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter is not in the SUBSCRIBE sent and it must not be in the NOTIFY to match that subscription.

Returns

- **resFE_INVALID_ARGUMENT:** The id is already used by another subscription in this object or the event is unknown.
- **resFE_INVALID_STATE:** The manager of the service was not set, Clear has been called on the context or the attached user agent service is not properly configured (e.g.: contact list is empty).
- **resFE_SIPCORE_PACKET_BLOCKED:** One of the service synchronously blocked the packet. No additional event will be reported.
- **resFE_FAIL:** The SUBSCRIBE request could not be sent.
- **resS_OK:** The SUBSCRIBE request is successfully sent.

Description

Creates a subscription by sending a SUBSCRIBE request. That subscription works exactly as an explicit subscription created by the CreateSubscription (see page 598) method. The status of this request is given to the application through the ISipSubscriberMgr (see page 590) interface. Note that the initial NOTIFY message for this subscription can be received before the response to the SUBSCRIBE request.

The expiration time is a suggestion to the remote peer. The actual expiration time can be lowered by the peer. If the subscription is still active at the threshold time, before the expiration arrives, the EvExpiring event is triggered for that subscription. The application can then decide to continue the subscription by calling Refresh (see page 599)() or to terminate it by calling Terminate (see page 602)(). If none of these actions are taken by the application before the subscription expires, the EvExpired event is called for that subscription.

See Also

Refresh (see page 599), Terminate (see page 602), AddEvent (see page 597)

3.7.1.33.1.9 - ISipSubscriberSvc::Terminate Method

Terminates an active subscription.

C++

```
virtual mxt_result Terminate(IN const CString& rstrEvent, IN const CString& rstrId, IN mxt_opaque
opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT
ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrEvent	The event type, e.g., "message-summary".
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL, which means that no extra headers are added. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	A pointer to the description of the message-body that should accompany the SUBSCRIBE request to be sent. It can be NULL, which means that no message-body is added to the request. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
strId	The "id" parameter that is put in the Event header field of the SUBSCRIBE request sent. It should also be in the Event header field of the NOTIFY received for that subscription. When the empty string is used, the id parameter is not in the SUBSCRIBE sent and it must not be in the NOTIFY to match that subscription.

Returns

- resFE_INVALID_ARGUMENT: The id does not correspond to any subscription in this object or the event is unknown.
- resFE_INVALID_STATE: The manager of the service was not set or Clear has been called on the context.
- resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
- resFE_FAIL: The SUBSCRIBE request could not be sent.
- resS_OK: The SUBSCRIBE request is successfully sent.

Description

Terminates the subscription corresponding to the id in parameter. If the subscription is an implicit subscription, it only updates its state to automatically refuse NOTIFY for this subscription. Otherwise, it sends a SUBSCRIBE request with the Expires header set to 0. The subscription is considered terminated when a NOTIFY with status terminated is received or 32 seconds after the SUBSCRIBE receives its final response.

See Also

Subscribe (see page 601), CreateSubscription (see page 598), AddEvent (see page 597)

3.7.1.33.2 - Enumerations

3.7.1.33.2.1 - ISipSubscriberSvc::EType Enumeration

```
enum EType {
    eIMPLICIT,
    eEXPLICIT
};
```

Description

Indicate how the subscription is created.

See Also

CreateSubscription (see page 598)

Members

Members	Description
eIMPLICIT	The subscription is established by other means than SIP
eEXPLICIT	The subscription has been established by SIP signalling but not via a SUBSCRIBE request

3.7.1.34 - ISipTransactionCompletionMgr Class New in 4.1.4

Class Hierarchy

ISipTransactionCompletionMgr

C++

```
class ISipTransactionCompletionMgr;
```

Description

The transaction completion manager is the interface through which the ISipTransactionCompletionSvc (see page 605) reports its events to the application. It is through this interface that the M5T SIP-UA SAFE informs the application that a transaction is completed.

Location

SipUserAgent/ISipTransactionCompletionMgr.h

See Also

ISipTransactionCompletionSvc (see page 605)

Methods

Method	Description
EvTransactionCompleted (see page 604)	Notifies that all transactions for a specific request type have been completed.

Legend

	Method
	abstract

3.7.1.34.1 - Methods

3.7.1.34.1.1 - ISipTransactionCompletionMgr::EvTransactionCompleted Method

Notifies that all transactions for a specific request type have been completed.

C++

```
virtual void EvTransactionCompleted(IN const CString& rstrMethod, IN ISipContext* pContext) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrMethod	The method of the request for which all transactions were completed.
IN ISipContext* pContext	Pointer to the context associated with the transaction. Cannot be NULL.

Description

Reports that all transactions of the method type identified by rstrMethod are terminated on the context. This means that the stack is now ready to send another request that may have been buffered by the application.

This event is only reported when it is okay for the application to send a new request. When a request is sent or received and a final response that terminates the transaction is received or sent, the transaction count becomes zero and an asynchronous message is posted (this will eventually report EvTransactionCompleted when the message is processed). When the asynchronous message is processed, if the transaction count is still zero, then the event is reported. For some reasons, it is possible that the transaction count is not zero when the asynchronous message is processed. A request could either be received or sent while there is an asynchronous message in the queue of the transaction completion service. That explains why the event is not always reported after a transaction has terminated.

For example, an UPDATE request is sent and a final response is received, then an asynchronous message is posted. Before the asynchronous is processed, another UPDATE is received (the transaction count is incremented). Hence when the asynchronous message is processed, the event is not reported.

See Also

[ISipTransactionCompletionSvc](#) (see page 605)

3.7.1.35 - ISipTransactionCompletionSvc Class New in 4.1.4

Class Hierarchy



C++

```
class ISipTransactionCompletionSvc : public IEComUnknown;
```

Description

This service monitors ongoing transactions and indexes them by their type of request. When all client and server transactions of a specific request type are terminated, this service reports an event through its manager interface. The application then knows that it can send a new request.

This service also allows to configure how the generic service behaves when trying to initiate multiple transactions of the same request type in parallel. By default, the generic service is able to manage multiple parallel client and server transactions for the same type of request. By using `EnableParallelTransactions` (see page 606) ("method", false), the application prevents the generic service from creating a client transaction for a specific request type when there is already an existing transaction (client or server) for this request type. In other words, the default behaviour is to allow parallel transactions for a specific request type, while calling `EnableParallelTransactions` (see page 606) with false forces serializing the client transactions at the application level.

This service generates events to the application through the `ISipTransactionCompletionMgr` (see page 604) interface.

This service must be one of the last services to be attached to a context in order to properly work. Only the following services can be attached to a context after the `ISipTransactionCompletedSvc`: `ISipStatisticsSvc` (see page 123), `ISipOutputControllingSvc`, and `ISipConnectionBlacklistSvc` (see page 84).

The ISipTransactionCompletionSvc is an ECOM object

The `ISipTransactionCompletionSvc` is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_ISipTransactionCompletionSvc

Interface Id: IID_ISipTransactionCompletionSvc

A user can query the `ISipContext` (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the `ISipContext` (see page 23) through the same mean.

Location

`SipUserAgent/ISipTransactionCompletionSvc.h`

See Also

[ISipTransactionCompletionMgr](#) (see page 604)

Methods

Method	Description
◆ A <code>CanEstablishNewTransaction</code> (see page 606)	Tells if a new transaction can be established.
◆ A <code>EnableParallelTransactions</code> (see page 606)	Sets if parallel transactions are allowed for the specified method.
◆ A <code>SetManager</code> (see page 606)	Sets the manager to notify of transaction completion.

Legend

◆	Method
◆ A	abstract

3.7.1.35.1 - Methods

3.7.1.35.1.1 - ISipTransactionCompletionSvc::CanEstablishNewTransaction Method

Tells if a new transaction can be established.

C++

```
virtual bool CanEstablishNewTransaction(IN const CString& rstrMethod) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrMethod	The method that will create the transaction.

Returns

true if a new transaction based on the request strMethod can be established.

Description

Returns true if a new transaction based on the request strMethod is allowed to be created. A new transaction will be denied to be created only if there is already an ongoing transaction and that parallel transactions for this request type is set to false.

3.7.1.35.1.2 - ISipTransactionCompletionSvc::EnableParallelTransactions Method

Sets if parallel transactions are allowed for the specified method.

C++

```
virtual void EnableParallelTransactions(IN const CString& rstrMethod, IN bool bEnable) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrMethod IN bool bEnable	The method type to configure. • true to allow parallel transactions. • false to disallow parallel transactions and force serial processing.

Description

Sets if the passed strMethod can have multiple parallel transactions. By default, a method allows parallel transactions. Note that this setting applies only to the ISipGenericSvc (see page 481) on the current ISipContext (see page 23).

3.7.1.35.1.3 - ISipTransactionCompletionSvc::SetManager Method

Sets the manager to notify of transaction completion.

C++

```
virtual void SetManager(IN ISipTransactionCompletionMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipTransactionCompletionMgr* pMgr	Pointer to the manager to notify of the transaction completion.

Description

Sets the transaction completion manager that is notified when all transactions for a specific request type are completed.

3.7.1.36 - ISipTransferMgr07 Class

Class Hierarchy

```
ISipTransferMgr07
```

C++

```
class ISipTransferMgr07;
```

Description

The transfer manager 07 is the interface through which the transfer service 07 talks to the application. It is through this interface that the M5T SIP Stack notifies the application of the progress of a transfer.

The transfer service 07 can manage one transfer as transferee and one transfer as transferor at the same time.

The transfer service 07 does NOT respond to any valid request that are notified to the application by this manager, so the manager must answer them when it receives a request related event other than EvInvalidRequest (see page 608). The transfer service 07 does NOT send any request by itself, so the application must send the NOTIFY requests stating its progress to the transferor.

Location

SipUserAgent/ISipTransferMgr07.h

See Also

ISipTransferSvc07 (see page 614)

Methods

Method	Description
• A EvFinalReport (see page 607)	A final NOTIFY has been received.
• A EvFinalStatusRequired (see page 608)	The subscription is terminated so a final NOTIFY should be sent.
• A EvInvalidRequest (see page 608)	An invalid request has been received.
• A EvNotifyFailure (see page 609)	A failure response has been received for a NOTIFY request.
• A EvNotifyProgress (see page 610)	A provisional response for a NOTIFY request has been received.
• A EvNotifySuccess (see page 610)	A success response has been received for a NOTIFY request.
• A EvProgressReport (see page 611)	A non-final NOTIFY has been received.
• A EvReferFailure (see page 611)	A failure response has been received for the REFER request.
• A EvReferProgress (see page 612)	A provisional response has been received for the REFER request.
• A EvReferSuccess (see page 612)	A success response has been received for the REFER request.
• A EvReportingExpired (see page 613)	The subscription expired before a final NOTIFY has been received.
• A EvTransferred (see page 613)	A valid REFER request has been received.

Legend

•	Method
A	abstract

3.7.1.36.1 - Methods

3.7.1.36.1.1 - ISipTransferMgr07::EvFinalReport Method

A final NOTIFY has been received.

C++

```
virtual void EvFinalReport(IN ISipTransferSvc07* pSvc, IN ISipServerEventControl* pServerEventControl, IN const CSipStatusLine& rStatus, IN const CHeaderList* pContentHeaders, IN const CSipPacket& rNotifyRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.

IN ISipServerEventControl* pServerEventControl	The interface through which the application must send at least one response. If SendResponse is not called directly from the present method, a reference must be added on pServerEventControl. The added reference must be released after SendResponse is called.
IN const CSipStatusLine& rStatus	The status line found in the content of the NOTIFY request.
IN const CHeaderList* pContentHeaders	Headers that accompanied the status line in the content of the NOTIFY. It can be NULL, which means that there were no such headers. Note that the application does not have ownership of this pointer and the header list must be copied if the application intends to use it after the method returned.
IN const CSipPacket& rNotifyRequest	The NOTIFY request.

Description

Notifies the application that a valid final NOTIFY request has been received.

A NOTIFY is final when the value of the Subscription-State header is "terminated".

The application should accept the NOTIFY request by replying 200 OK through pServerEventControl.

The transfer is considered terminated only when a final success response (2xx) is sent to this NOTIFY. If no final NOTIFY is responded 2xx, then the transfer is considered terminated when EvReportingExpired (see page 613) is called.

Note that if this final NOTIFY is refused (a failure response is sent by the application), it is not guaranteed that the transferee will still send NOTIFY requests. In this case, EvReportingExpired (see page 613) will eventually be called.

See Also

EvProgressReport (see page 611), EvReportingExpired (see page 613), ISipTransferSvc07 (see page 614)

3.7.1.36.1.2 - ISipTransferMgr07::EvFinalStatusRequired Method

The subscription is terminated so a final NOTIFY should be sent.

C++

```
virtual void EvFinalStatusRequired(IN ISipTransferSvc07* pSvc, IN const CSipPacket* pSubscribe) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN const CSipPacket* pSubscribe	The SUBSCRIBE request received that terminated the subscription. It is NULL if this event was called for an expiration of the subscription.

Description

The subscription created by the REFER request has terminated.

A SUBSCRIBE request may have been received to terminate the subscription.

The other possibility is that the subscription is expired. This happens only for subscriptions created by accepting a REFER request through EvTransferred (see page 613) for which no final NOTIFY request has been sent.

In both cases, a final NOTIFY request should be sent to the transferor (either with ISipTransferSvc07::ReportFinalStatus (see page 616), ISipTransferSvc07::ReportFailure (see page 616), or ISipTransferSvc07::ReportRefusal (see page 619)) (RFC 3265, section 3.1.4.3 for the SUBSCRIBE and section 3.1.6.4 for the expiration).

See Also

EvTransferred (see page 613), ISipTransferSvc07::ReportFinalStatus (see page 616), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportRefusal (see page 619)

3.7.1.36.1.3 - ISipTransferMgr07::EvInvalidRequest Method

An invalid request has been received.

C++

```
virtual void EvInvalidRequest(IN ISipTransferSvc07* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest, IN mxt_result resError) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfers on the context.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rRequest	The invalid request.
IN mxt_result resError	resFE_REQUEST_PENDING: The REFER request has been received when the transfer service 07 is already processing a transfer as a transferee. A 491 response has automatically been sent by the transfer service 07.
resFE_UNSUPPORTED_URI_SCHEME	The REFER request has been received with an unsupported URI scheme. A 488 "Unsupported Refer-To URI Scheme" response has automatically been sent by the transfer service 07.
resFE_MISSING_HEADER	The REFER request is missing a valid Refer-To header. A "400 Bad Request" response has already been sent.
resFE_UNKNOWN_EVENT	The SUBSCRIBE request is missing an Event header or the event type is not "refer". A "489 Bad Event" response has already been sent. The Event header was missing or for an event type other than "refer". A "489 Bad Event" response has already been sent.
resFE_UNKNOWN_SUBSCRIPTION	The SUBSCRIBE request did not correspond to any REFER sent by this service. A "481 Subscription does not exist" response has already been sent. The NOTIFY does not correspond to a REFER sent by this service. A "481 Subscription does not exist" response has already been sent.
resFE_EXPIRED_SUBSCRIPTION	The SUBSCRIBE has been received after a NOTIFY with "subscription-state" "terminated" was sent or after the expiration time. A "481 Subscription does not exist" response has already been sent. The NOTIFY has been received after another NOTIFY with "subscription-state" "terminated" or after the expiration time. A "481 Subscription does not exist" response has already been sent.
resFE_INVALID_CONTENT	The received NOTIFY did not contain a valid "message/sipfrag".

Description

Warns the application that an invalid request has been received. The request was automatically responded with an error response by the service. Usually, the application has no action to take other than reporting this invalid packet if wanted.

Take note that if the manager of the service is not set, all the requests are responded with a 500 response.

3.7.1.36.1.4 - ISipTransferMgr07::EvNotifyFailure Method

A failure response has been received for a NOTIFY request.

C++

```
virtual void EvNotifyFailure(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN
const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelIssueRequest on it. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The final failure response to a NOTIFY.

Description

A final failure response to a NOTIFY has been received.

NOTIFY requests are sent with ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), or ISipTransferSvc07::ReportFinalStatus (see page 616).

See Also

ISipTransferSvc07 (see page 614), ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), ISipTransferSvc07::ReportFinalStatus (see page 616).

3.7.1.36.1.5 - ISipTransferMgr07::EvNotifyProgress Method

A provisional response for a NOTIFY request has been received.

C++

```
virtual void EvNotifyProgress(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN
const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The provisional response to a NOTIFY.

Description

A provisional response to a NOTIFY has been received.

NOTIFY requests are sent with ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), or ISipTransferSvc07::ReportFinalStatus (see page 616).

See Also

ISipTransferSvc07 (see page 614), ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), ISipTransferSvc07::ReportFinalStatus (see page 616).

3.7.1.36.1.6 - ISipTransferMgr07::EvNotifySuccess Method

A success response has been received for a NOTIFY request.

C++

```
virtual void EvNotifySuccess(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN
const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelissueRequest on it. Note however that RelissueRequest is usually called to retry failed requests instead of successful ones. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The final success response to a NOTIFY.

Description

A final success response to a NOTIFY has been received.

NOTIFY requests are sent with ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), or ISipTransferSvc07::ReportFinalStatus (see page 616).

See Also

ISipTransferSvc07 (see page 614), ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportProgress (see page 618), ISipTransferSvc07::ReportFinalStatus (see page 616).

3.7.1.36.1.7 - ISipTransferMgr07::EvProgressReport Method

A non-final NOTIFY has been received.

C++

```
virtual void EvProgressReport(IN ISipTransferSvc07* pSvc, IN ISipServerEventControl* pServerEventControl, IN ISipReferrerMgr::EState eState, IN const CSipStatusLine& rStatus, IN const CHeaderList* pContentHeaders, IN const CSipPacket& rNotifyRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipServerEventControl* pServerEventControl	The interface through which the application must send at least one response. If SendResponse is not called directly from the present method, a reference must be added on pServerEventControl. The added reference must be released after SendResponse is called.
IN ISipReferrerMgr::EState eState	The state of the subscription. It is either ePENDING or eACTIVE. This parameter is calculated from the Subscription-State header of the NOTIFY request received.
IN const CSipStatusLine& rStatus	The status line found in the content of the NOTIFY request.
IN const CHeaderList* pContentHeaders	Headers that accompanied the status line in the content of the NOTIFY. It can be NULL, which means that there were no such headers. Note that the application does not have ownership of this pointer and the header list must be copied if the application intends to use it after the method returned.
IN const CSipPacket& rNotifyRequest	The NOTIFY request.

Description

Informs the application that a valid non-final NOTIFY request has been received.

A NOTIFY is final when the value of the Subscription-State header is "terminated".

The application should accept the NOTIFY request by replying 200 OK through pServerEventControl.

See Also

EvFinalReport (see page 607), ISipTransferSvc07 (see page 614)

3.7.1.36.1.8 - ISipTransferMgr07::EvReferFailure Method

A failure response has been received for the REFER request.

C++

```
virtual void EvReferFailure(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent, ClearClientEvents, or RelissueRequest on it. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The failure response to the REFER.

Description

A failure response to the REFER has been received.

The manager must be prepared to receive `EvProgressReport` (see page 611) or `EvFinalReport` (see page 607) before this one is received (RFC 3515, section 2.4.4).

REFER requests are sent with `ISipTransferSvc07::Transfer` (see page 621).

See Also

`ISipTransferSvc07` (see page 614), `ISipTransferSvc07::Transfer` (see page 621).

3.7.1.36.1.9 - `ISipTransferMgr07::EvReferProgress` Method

A provisional response has been received for the REFER request.

C++

```
virtual void EvReferProgress(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either <code>CallNextEvent</code> or <code>ClearClientEvents</code> on it. If one of these methods is not called directly from the present method, a reference must be added on <code>pClientEventControl</code> . The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The provisional response to the REFER.

Description

A provisional response to the REFER has been received.

REFER requests are sent with `ISipTransferSvc07::Transfer` (see page 621).

See Also

`ISipTransferSvc07` (see page 614), `ISipTransferSvc07::Transfer` (see page 621).

3.7.1.36.1.10 - `ISipTransferMgr07::EvReferSuccess` Method

A success response has been received for the REFER request.

C++

```
virtual void EvReferSuccess(IN ISipTransferSvc07* pSvc, IN ISipClientEventControl* pClientEventControl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.
IN ISipClientEventControl* pClientEventControl	The client event control interface for this transaction. It cannot be NULL. The application must call either <code>CallNextEvent</code> , <code>ClearClientEvents</code> , or <code>RelissueRequest</code> on it. Note however that <code>RelissueRequest</code> is usually called to retry failed requests instead of successful ones. If one of these methods is not called directly from the present method, a reference must be added on <code>pClientEventControl</code> . The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The success response to the REFER.

Description

A success response to the REFER has been received.

The manager must be prepared to receive `EvProgressReport` (see page 611) or `EvFinalReport` (see page 607). These events may be received before this one is received (RFC 3515, section 2.4.4).

REFER requests are sent with `ISipTransferSvc07::Transfer` (see page 621).

See Also

ISipTransferSvc07 (see page 614), ISipTransferSvc07::Transfer (see page 621).

3.7.1.36.1.11 - ISipTransferMgr07::EvReportingExpired Method

The subscription expired before a final NOTIFY has been received.

C++

```
virtual void EvReportingExpired(IN ISipTransferSvc07* pSvc) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer.

Description

Notifies the application that a subscription is expired. There is no action to take by the application.

A NOTIFY request is no longer received so the transfer is considered terminated.

3.7.1.36.1.12 - ISipTransferMgr07::EvTransferred Method

A valid REFER request has been received.

C++

```
virtual void EvTransferred(IN ISipTransferSvc07* pSvc, IN ISipServerEventControl* pServerEventControl, IN const CNameAddr& rTarget, IN const CSipPacket& rReferRequest) = 0;
```

Parameters

Parameters	Description
IN ISipTransferSvc07* pSvc	The service managing the transfer initiated with the REFER request.
IN ISipServerEventControl* pServerEventControl	The interface through which the application must send at least one response. If SendResponse is not called directly from the present method, a reference must be added on pServerEventControl. The added reference must be released after SendResponse is called.
IN const CNameAddr& rTarget	The transfer target URL. This is the URL found in the Refer-To header.
IN const CSipPacket& rReferRequest	The REFER request.

Description

Notifies the application that the remote user-agent (the transferor) wants the local user-agent (acting as a transferee) to contact the agent at the target URL (the transfer target).

When this method is called, the REFER request has a valid Refer-To header containing a supported URI and the service does not manage any transfer as a transferee.

If the REFER request is answered with a 2xx response, the application MUST immediately send a NOTIFY to the transferor (RFC 3515, section 2.4.4). The first NOTIFY request sent for the REFER may be a "100 Trying", which may be sent by using ISipTransferSvc07::ReportPending (see page 617).

Since the REFER request contains a Refer-To header and the target URI has a supported scheme, the application should answer a 202 response to the REFER request, send a NOTIFY with a "100 Trying" payload (by using ISipTransferSvc07::ReportPending (see page 617)), and then see if it accepts to do the transfer (user input should be used, either through configuration or in real time). If the user accepts, it should create a new context to rTarget and call ISipSessionSvc::Invite (see page 581) method on it.

If, after having sent a 202 response to the REFER request, the application does not accept the transfer, it should send a final NOTIFY request containing a payload of "603 Declined" to the transferor by using ISipTransferSvc07::ReportRefusal (see page 619).

If the transfer was tried but failed, the application should use ISipTransferSvc07::ReportFailure (see page 616) to send a final NOTIFY to the transferor.

Note that a final NOTIFY MUST be sent for every REFER accepted (draft-ietf-sipping-cc-transfer-03, section 3, requirement 3) so either ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), or ISipTransferSvc07::ReportFinalStatus (see page 616) must be called once for every REFER accepted (responded 2xx).

If the application does not want to continue the subscription (even before the first NOTIFY is sent), it should call ReportFinalStatus to send a final NOTIFY.

See Also

ISipTransferSvc07::ReportPending (see page 617), ISipTransferSvc07::ReportFailure (see page 616), ISipTransferSvc07::ReportRefusal (see page 619), ISipTransferSvc07::ReportFinalStatus (see page 616), ISipTransferSvc07 (see page 614).

3.7.1.37 - ISipTransferSvc07 Class

Class Hierarchy



C++

```
class ISipTransferSvc07 : public IEComUnknown;
```

Description

The transfer service 07 implements the RFC 3515 (REFER Method) and the draft-ietf-sipping-cc-transfer-03 for transfer. It is also based on RFC 3265 for event notification.

The transfer service 07 does NOT manage the Referred-By header and related content.

The transfer service is used to transfer an existing session to another user using the REFER method. It can be used as a transferor and as a transferee.

The service can manage one transfer as a transferor and as a transferee at the same time. If a REFER request is received while the service manages a transfer as a transferee, it is automatically refused with a 491 "Request Pending" response.

It can only be used to transfer to other SIP targets and only to trigger INVITE requests. The ISipReferrerSvc (see page 542) (or ISipGenericSvc (see page 481)) can be used to send a REFER that triggers a request other than an INVITE or that has a target other than a SIP-URL.

If the Transferee is using a GRUU (Globally Routable User Agent, see draft-ietf-sip-gruu-03), the REFER requests should be sent in a new dialog (other than the INVITE one). If not, the REFER requests should be sent inside the INVITE dialog.

The REFER request creates a subscription if it is accepted by the transferee. It is currently not possible to resubscribe to or unsubscribe from a subscription by sending a SUBSCRIBE request.

When the transfer service 07 acts as a transferor and receives a SUBSCRIBE corresponding to the REFER request with the Event value set to "refer", it automatically accepts it with a 200 OK response. If the SUBSCRIBE request is an unsubscribe request, the service calls EvFinalStatusRequired on its manager.

Note that after creating this service, the application should call SetManager (see page 620) on it because otherwise no request can be sent with it and the service refuses all the requests it receives with a 500 "Internal Server Error" response.

When acting as a transferee, the length of the subscription must be set in the first NOTIFY sent by using the uExpirationInSec parameter of the choose method (ReportPending (see page 617) or ReportProgress (see page 618)).

The transfer service 07 does NOT respond to REFER and NOTIFY requests automatically when they are acceptable. The service does not send NOTIFY requests automatically. All these actions are the application's responsibility.

The ONLY thing this service does automatically is respond to SUBSCRIBE requests and non-acceptable requests.

This service generates events to the application through the ISipTransferMgr interface.

The ISipTransferSvc07 is an ECOM object

The ISipTransferSvc07 is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipTransferSvc07

Interface Id: IID_ISipTransferSvc07

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Notes

This class follows the rules of draft-ietf-sipping-cc-transfer-03. The class enumeration is incremental. Since the last corresponding version in v3.6 was 06, this one is called 07.

Location

SipUserAgent/ISipTransferSvc07.h

See Also

ISipTransferMgr07 (see page 607)

Methods

Method	Description
ConfigureNotifyIdParameterUsage (see page 615)	Configures whether or not the stack will use the "id" parameter in the Event header of NOTIFY requests for first REFER request.
ReportFailure (see page 616)	Notifies the transferor that the transfer failed.
ReportFinalStatus (see page 616)	Notifies the transferor of the transfer progress and informs it that this is the last information it will receive on the progress.
ReportPending (see page 617)	Notifies the transferor that the transfer is tried.
ReportProgress (see page 618)	Notifies the transferor of the transfer progress.
ReportRefusal (see page 619)	Notifies the transferor that the transfer has been declined.
SetManager (see page 620)	Configures the manager associated with this service.
Transfer (see page 621)	Sends a REFER request to initiate a transfer.

Legend

Method	
abstract	

3.7.1.37.1 - Methods**3.7.1.37.1.1 - ISipTransferSvc07::ConfigureNotifyIdParameterUsage Method**

Configures whether or not the stack will use the "id" parameter in the Event header of NOTIFY requests for first REFER request.

C++

```
virtual void ConfigureNotifyIdParameterUsage(IN ISipRefereeSvc::EIdParameterUsage eIdParamUsage) = 0;
```

Parameters

Parameters	Description
IN ISipRefereeSvc::EIdParameterUsage eIdParamUsage	<ul style="list-style-type: none"> eID_PARAM_ALWAYS_PRESENT: The id parameter is added to all NOTIFYs. This is the default behaviour. eID_PARAM_ABSENT_FOR_FIRST_REFERER: The id parameter in the NOTIFYs to the first REFER are not set.

Description

Configures whether or not the stack will use the "id" parameter in the Event header of the NOTIFY it sends after a subscription is created by receiving a REFER. This is used only for the NOTIFY(s) associated with the first REFER received by this service.

RFC 3515 states that the NOTIFY(s) associated with the first REFER sent on a dialog MAY contain an "id" parameter in the Event header, and the NOTIFY(s) associated with any additional REFER after the first one MUST contain this parameter. The default behavior of the stack is to always include the parameter for all NOTIFY(s) it sends, however this can sometimes cause interoperability issues. An application can use this method to configure whether or not the stack will include the "id" parameter for the NOTIFY(s) associated with the first REFER it has received. All NOTIFYs associated with subsequent REFERs will always contain the "id" parameter in the Event header.

See Also

[ISipRefereeSvc](#) (see page 531)

3.7.1.37.1.2 - ISipTransferSvc07::ReportFailure Method

Notifies the transferor that the transfer failed.

C++

```
virtual mxt_result ReportFailure(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, OUT
ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE: No transfer is currently managed as a transferee or the original REFER request has not been accepted yet.
 resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.
 resFE_FAIL: The NOTIFY request could not be sent.
 resS_OK: If the final NOTIFY request has been correctly sent.

Description

This method is used to send a final NOTIFY request to the remote host connected with the parent context with a payload of "503 Service Unavailable". This method should be called when the transfer failed (e.g.: The INVITE was refused by the target, it timed up, or the user got tired of waiting for a final response).

If the transfer is declined by the application before it is attempted (e.g.: The user has refused to follow the REFER request it received) and a 2xx response has been sent for the REFER, the application should send a payload of "603 Declined" (RFC 3515, section 2.4.5) by using ReportRefusal (see page 619).

The NOTIFY sent by this method is a final NOTIFY so no other NOTIFY request may be sent for the transfer. The NOTIFY contains a Subscription-State header set to "terminated" and a reason parameter of "noresource".

Note that a final NOTIFY MUST be sent for every REFER accepted (draft-ietf-sipping-cc-transfer-03, section 3, requirement 3) so either ReportRefusal (see page 619), ReportFinalStatus (see page 616), or this method must be called once for every REFER accepted (responded 2xx).

IMPORTANT: This method does NOT send a response for the REFER request, the application must send it on its own.

See Also

[ISipTransferMgr07](#) (see page 607)

3.7.1.37.1.3 - ISipTransferSvc07::ReportFinalStatus Method

Notifies the transferor of the transfer progress and informs it that this is the last information it will receive on the progress.

C++

```
virtual mxt_result ReportFinalStatus(IN mxt_opaque opqTransaction, IN const CSipStatusLine& rContentStatusLine,
IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.

IN const CSipStatusLine& rContentStatusLine	The status-line to put in the content of the NOTIFY request. It is the actual referral status. Note that this response can be either final or provisional. The fact that this NOTIFY is final simply indicates to the transferor that the transferee will no longer inform it of the refer request progress. When this method is used to report a progress received from the transfer target (on the network), you can take the status line of the packet by using CSipPacket::GetStatusLine.
IN TO CHeaderList* pExtraHeaders OUT ISipClientTransaction*& rpTransaction	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN. Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE:

- No transfer is currently managed as a transferee or the original REFER request has not been accepted yet.
- A transfer is managed as a transferee but the manager is NULL.

This can happen when Clear was called on the context.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The NOTIFY request could not be sent.

resS_OK: If the final NOTIFY request has been correctly sent.

Description

This method is used to send a NOTIFY request to the remote host connected with the parent context with a payload corresponding to rContentStatusLine.

The Subscription-State header of the NOTIFY sent is set to "terminated" and a reason parameter of "noresource".

When other responses are expected from the transfer target, ReportProgress (see page 618) should be used instead of this method.

The NOTIFY sent by this method is a final NOTIFY so no other NOTIFY request may be sent for the transfer.

Note that a final NOTIFY MUST be sent for every REFER accepted (draft-ietf-sipping-cc-transfer-03, section 3, requirement 3) so either ReportRefusal (see page 619), ReportFailure (see page 616), or this method must be called once for every REFER accepted (responded 2xx).

RFC 3515 states that a final NOTIFY MUST indicate the subscription has been "terminated" with a reason of "noresource", except when the REFER was accepted with a 202 but approval to follow the reference was then refused (section 2.4.7). In the second case, ReportRefusal (see page 619) should be used instead of ReportFinalStatus.

IMPORTANT: This method does NOT send a response for the REFER request, the application must send it on its own.

See Also

ISipTransferMgr07 (see page 607), CSipPacket::GetStatusLine

3.7.1.37.1.4 - ISipTransferSvc07::ReportPending Method

Notifies the transferor that the transfer is tried.

C++

```
virtual mxt_result ReportPending(IN mxt_opaque opqTransaction, IN unsigned int uExpirationInSec, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.

IN unsigned int uExpirationInSec	The expiration time of the subscription initiated with the REFER request. This expiration should be set to the maximum time expected to obtain a final status for the INVITE request. The value of uExpirationInSec cannot be increased after the first NOTIFY request is sent. Only the sender of the REFER can extend the subscription by sending a SUBSCRIBE request.
IN TO CHeaderList* pExtraHeaders OUT ISipClientTransaction*& rpTransaction	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN. Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE:

- No transfer is currently managed as a transferee or the original REFER request has not been accepted yet.

- A transfer is managed as a transferee but the manager is NULL.

This can happen when Clear was called on the context.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The NOTIFY request could not be sent.

resS_OK: If the final NOTIFY request has been correctly sent.

Description

This method is used to send a NOTIFY request to the remote host connected with the parent context with a payload of "100 Trying". This method should be called when the a 2xx response has been responded to the REFER request initiating the transfer.

Note that a NOTIFY request MUST immediately be sent to the transferor when a REFER request is accepted (2xx) (RFC 3515, section 2.4.4).

The Subscription-State header of the NOTIFY is set to "pending".

See Also

ISipTransferMgr07 (see page 607)

3.7.1.37.1.5 - ISipTransferSvc07::ReportProgress Method

Notifies the transferor of the transfer progress.

C++

```
virtual mxt_result ReportProgress(IN mxt_opaque opqTransaction, IN unsigned int uExpirationInSec, IN const CSipStatusLine& rContentStatusLine, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction IN unsigned int uExpirationInSec	Application data to associate with this transaction. This is opaque to the service. The expiration time of the subscription initiated with the REFER request. This expiration should be set to the maximum time expected to obtain a final status for the INVITE request. The value of uExpirationInSec cannot be increased after the first NOTIFY request is sent. Only the sender of the REFER can extend the subscription by sending a SUBSCRIBE request. This value is put in the "expires" parameter of the "Subscription-State" header of the NOTIFY request sent. When 0, the current remaining expiration for that subscription is used. The application should not use 0 the first time it calls this method (or ReportProgress) since the initial expiration is set really high so the application can decrease it to what it wants.

IN const CSipStatusLine& rContentStatusLine	The status-line to put in the content of the NOTIFY request. It is the actual referral status. Note that it can be a final response when other responses are still expected (because the application will retry the referred request for instance). When no other responses are expected, ReportFinalStatus (see page 616) should be used instead. When this method is used to report a progress received from the transfer target, you can take the status line of the packet by using CSipPacket::GetStatusLine.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE:

- No transfer is currently managed as a transferee or the original REFER request has not been accepted yet.

- A transfer is managed as a transferee but the manager is NULL. This can happen when Clear was called on the context.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The NOTIFY request could not be sent.

resS_OK: If the final NOTIFY request has been correctly sent.

Description

This method is used to send a NOTIFY request to the remote host connected with the parent context with a payload corresponding to the status line in parameter. This method should be called when responses (provisional or final) are received for the INVITE triggered by the reception of a REFER, and other responses will still be received for the INVITE (because the application will retry the INVITE for instance in case of a final response).

The Subscription-State header is set to "active" with this method, meaning that the transferor will receive other NOTIFY requests for further progress.

When no other response is expected from the transfer target, ReportFinalStatus (see page 616) should be used instead of this method.

IMPORTANT: This method does NOT send a response for the REFER request, the application must send it on its own.

See Also

ReportFinalStatus (see page 616), ISipTransferMgr07 (see page 607), CSipPacket::GetStatusLine

3.7.1.37.1.6 - ISipTransferSvc07::ReportRefusal Method

Notifies the transferor that the transfer has been declined.

C++

```
virtual mxt_result ReportRefusal(IN mxt_opaque opqTransaction, IN ISipNotifierSvc::EReason eReason, IN unsigned int uRetryAfterSec, IN TO CHeaderList* pExtraHeaders, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN ISipNotifierSvc::EReason eReason	The "reason" parameter to put in the "Subscription-State" header. When the value is eNO_REASON, the parameter is not put in the "Subscription-State" header. This reason can be used to tell if the REFER request can be re-attempted and why it was refused. Take note that other methods reporting status to the transferor use the eNO_RESOURCE value.
IN unsigned int uRetryAfterSec	The number of seconds to put in the "retry-after" parameter of the "Subscription-State" header. When the value is 0, no "retry-after" parameter is put. Note that RFC 3265 defines semantics for this parameter only if the eReason parameter has the value eNO_REASON, ePROBATION, or eGIVEUP.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN.

OUT ISipClientTransaction*& rpTransaction

Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE:

- No transfer is currently managed as a transferee or the original REFER request has not been accepted yet.

- A transfer is managed as a transferee but the manager is NULL.

This can happen when Clear was called on the context.

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The NOTIFY request could not be sent.

resS_OK: If the final NOTIFY request has been correctly sent.

Description

This method is used to send a final NOTIFY request to the remote host connected with the parent context with a payload of "603 Decline". This method should be called when the application knows that the transfer must be refused after a 202 response has been sent to the REFER (e.g.: The user decided not to accept it).

If the transfer was attempted but failed, ReportFailure (see page 616) should be used instead.

The NOTIFY sent by this method is a final NOTIFY so no other NOTIFY request may be sent for the transfer. The NOTIFY contains a Subscription-State header set to "terminated" and a reason parameter depending on eReason.

Note that a final NOTIFY MUST be sent for every REFER accepted (draft-ietf-sipping-cc-transfer-03, section 3, requirement 3) so either ReportFailure (see page 616), ReportFinalStatus (see page 616), or this method must be called once for every REFER accepted (responded 2xx).

IMPORTANT: This method does NOT send a response for the REFER request method, the application must send it on its own.

See Also

ISipTransferMgr07 (see page 607)

3.7.1.37.1.7 - ISipTransferSvc07::SetManager Method

Configures the manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipTransferMgr07* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipTransferMgr07* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT: pMgr is NULL. The manager is not changed in this case.

resS_OK: Otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.37.1.8 - Transfer

3.7.1.37.1.8.1 - ISipTransferSvc07::Transfer Method

Sends a REFER request to initiate a transfer.

C++

```
virtual mxt_result Transfer(IN mxt_opaque opqTransaction, IN const CNameAddr& rTarget, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN const CNameAddr& rTarget	The URL where the remote user-agent should send the request. It is put in the Refer-To header of the REFER request.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Information about the payload to send with the request. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.

Returns

resFE_INVALID_STATE:

- If this service has no manager.
- The service already manages a transfer as a transferor.

resFE_INVALID_ARGUMENT:

- rTarget is not a SIP-URI or SIPS-URI.
- rTarget is a SIP-URI or a SIPS-URI but contains a method-param with a value other than "INVITE".

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The REFER could not be sent.

resS_OK: The REFER request has been successfully sent.

Description

This method is used to send a REFER request to the remote host connected with the parent context. Notification about the request's progress is received through the ISipTransferMgr07 (see page 607) interface.

This service can only manage one transfer at a time as a transferor. Calling this method while a transfer is managed as a transferor fails. A transfer is managed as a transferor when a REFER has been sent and not finally answered yet or the response was a 2xx and no final NOTIFY has been received for that transfer.

Note that draft-ietf-sipping-cc-transfer-04 states that if the Contact URI given by the transferee in the INVITE is a GRUU, the transfer should be done on a new dialog. It is the caller's responsibility to create the new context and to call this method on the new context.

See Also

ISipTransferMgr07 (see page 607)

3.7.1.37.1.8.2 - ISipTransferSvc07::Transfer Method

Sends a REFER request to initiate an attended transfer.

C++

```
virtual mxt_result Transfer(IN mxt_opaque opqTransaction, IN const ISipUserAgentSvc* pUaDialog, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction, IN
```

```
bool bUseRemoteAddrInReferTo = true) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN const ISipUserAgentSvc* pUaDialog	The ISipUserAgentSvc (see page 639) of context of consultation in case of an attended transfer. It cannot be NULL.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Information about the payload to send with the request. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction. This interface is a reference counted object and a reference is counted for that parameter. The application is responsible for that reference.
IN bool bUseRemoteAddrInReferTo = true	When true the URL where the transferee should send the request will be taken from ISipUserAgentSvc::GetRemoteAddr (see page 647)(). It will be put in the Refer-To header of the REFER request. If the URI contains a Replaces header, it will be replaced by a Replaces header built by this method. This is the default behavior of the stack. When false the URL where the transferee should send the request will be taken from ISipUserAgentSvc::GetCurrentTarget (see page 644)() only if it is a SIP URI and put in the Refer-To header of the REFER request.

Returns

resFE_INVALID_STATE:

- If this service has no manager.
- The service already manages a transfer as a transferor.

resFE_INVALID_ARGUMENT:

- pUaDialog is NULL.
- pUaDialog is not in a dialog (nothing to replace).
- the remote address of pUaDialog is neither a SIP-URI or a SIPS-URI.
- the remote address of pUaDialog is a SIP-URI or SIPS-URI but contains a method-param with a value other than "INVITE".

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resFE_FAIL: The REFER could not be sent.

resS_OK: The REFER request has successfully been sent.

Description

This method is used to send a REFER request to the remote host connected with the parent context to begin an attended transfer. An attended transfer can be done by calling the transfer target before the REFER is sent to the transferee. When the INVITE is accepted by the transfer target, the call is then put on hold and this method is called on the context related to the transferee with the ISipUserAgentSvc (see page 639) related to the call to the transfer target.

Notification about the request's progress is received through the ISipTransferMgr07 (see page 607) interface.

This service can only manage one transfer at a time as a transferor. Calling this method while a transfer is managed as a transferor fails. A transfer is managed as a transferor when a REFER has been sent and not finally answered yet or the response was a 2xx and no final NOTIFY has been received for that transfer.

Note that draft-ietf-sipping-cc-transfer-04 states that if the Contact URI given by the transferee in the INVITE is a GRUU, the transfer

should be done on a new dialog. It is the caller's responsibility to create the new context and to call this method on the new context.

See Also

[ISipTransferMgr07](#) (see page 607)

3.7.1.38 - ISipUaAssertedIdentityMgr Class

Class Hierarchy

[ISipUaAssertedIdentityMgr](#)

C++

```
class ISipUaAssertedIdentityMgr;
```

Description

ISipUaAssertedIdentityMgr is the interface through which the ISipUaAssertedIdentitySvc (see page 628) reports events to the application.

The ISipUaAssertedIdentitySvc (see page 628) reports an event when a packet is received from an untrusted proxy or with at least a P-Asserted-Identity header.

If the packet has been received from an untrusted proxy, [EvUntrustedProxy](#) (see page 627) is called.

If the P-Asserted-Identity header(s) are valid, one of the [EvAssertedIdentity](#) (see page 624) events is called (the one called depends on whether the packet received is a request or a response).

If the P-Asserted-Identity header(s) are invalid, one of the [EvInvalidAssertedIdentity](#) (see page 625) events is called (still depending on whether the packet being a request or a response).

If FQDNs are used, the application SHOULD consider the [EvUntrustedProxy](#) (see page 627), [EvAssertedIdentity](#) (see page 624) and [EvInvalidAssertedIdentity](#) (see page 625) events of this manager with caution until the [EvTrustedProxyDnsResolutionCompleted](#) (see page 626) event has been reported at least once. As soon as the [EvTrustedProxyDnsResolutionCompleted](#) (see page 626) event is reported once, the service is in a reliable state. Before that event is reported, the DNS resolution is still in progress and the events might not be reliable.

In the case where ONLY IP addresses are used, the [EvTrustedProxyDnsResolutionCompleted](#) (see page 626) event will not be reported as no DNS resolution will be necessary. The service is thus in a reliable state right after the trusted proxy servers are set.

Location

[SipUserAgent/ISipUaAssertedIdentityMgr.h](#)

See Also

[ISipUaAssertedIdentitySvc](#) (see page 628)

Methods

Method	Description
  EvAssertedIdentity (see page 624)	A response has been received from the trusted proxy server with at least one P-Asserted-Identity header.
  EvInvalidAssertedIdentity (see page 625)	A response containing invalid P-Asserted-Identity has been received from the trusted proxy server.
  EvTrustedProxyDnsResolutionCompleted (see page 626)	The DNS resolution of the trusted proxy URI list is completed.
  EvUntrustedProxy (see page 627)	A response has been received from an untrusted proxy.

Legend

	Method
	abstract

3.7.1.38.1 - Methods

3.7.1.38.1.1 - EvAssertedIdentity

3.7.1.38.1.1.1 - ISipUaAssertedIdentityMgr::EvAssertedIdentity Method

A response has been received from the trusted proxy server with at least one P-Asserted-Identity header.

C++

```
virtual void EvAssertedIdentity(IN ISipUaAssertedIdentitySvc* pSvc, IN const CNameAddr* pSipIdentity, IN const CNameAddr* pTelIdentity, IN bool bPrivacyRequested, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identities UA side on this context.
IN const CNameAddr* pSipIdentity	The SIP or SIPS identity in a P-Asserted-Identity header present in the request. It MAY be NULL.
IN const CNameAddr* pTelIdentity	The telephone identity in a P-Asserted-Identity header present in the request. It MAY be NULL.
IN bool bPrivacyRequested	true if privacy was requested by the user for the Network Asserted Identity. In this case, the application MUST ensure that it does not forward the Network Asserted Identity outside the Trust Domain. false if privacy was not request by the user for the Network Asserted Identity.
IN ISipClientEventControl* pClientEventCtrl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The response containing the P-Asserted-Identity header(s).

Description

This event is called when a response is received from the trusted proxy server with valid P-Asserted-Identity header(s).

RFC 3325 states that a maximum of two headers can be present in a packet. One header contains a SIP or SIPS URI and the other contains a telephone URI. There may be one of the headers, both, or none. This event is not called if there is none.

The identity contained in the P-Asserted-Identity header(s) may be considered "privileged, or intrinsically more trustworthy than the From header field of a request" (RFC 3325, section 8, first paragraph).

Also, RFC 3325 states that "If a UA is part of the Trust Domain from which it received a message containing a P-Asserted-Identity header field, then it can use the value freely but it MUST ensure that it does not forward the information to any element that is not part of the Trust Domain, if the user has requested that asserted identity information be kept private." A user requests for asserted identity information to be kept private by inserting the 'id' privacy type into the Privacy header. If the privacy type is present, the bPrivacyRequested parameter is true.

Finally, "If a UA is not part of the Trust Domain from which it received a message containing a P-Asserted-Identity header field, then it can assume this information does not need to be kept private."

3.7.1.38.1.1.2 - ISipUaAssertedIdentityMgr::EvAssertedIdentity Method

A request has been received from the trusted proxy server with at least one P-Asserted-Identity header.

C++

```
virtual void EvAssertedIdentity(IN ISipUaAssertedIdentitySvc* pSvc, IN const CNameAddr* pSipIdentity, IN const CNameAddr* pTelIdentity, IN bool bPrivacyRequested, IN const CSipPacket& rRequest, INOUT mxt_opaque& rApplicationData) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identities UA side on this context.
IN const CNameAddr* pSipIdentity	The SIP or SIPS identity in a P-Asserted-Identity header present in the request. It MAY be NULL.
IN const CNameAddr* pTelIdentity	The telephone identity in a P-Asserted-Identity header present in the request. It MAY be NULL.

IN bool bPrivacyRequested	true if privacy was requested by the user for the Network Asserted Identity. In this case, the application MUST ensure that it does not forward the Network Asserted Identity outside the Trust Domain. false if privacy was not request by the user for the Network Asserted Identity.
IN const CSipPacket& rRequest	The request containing the P-Asserted-Identity header(s).
INOUT mxt_opaque& rApplicationData	Application data opaque to the service. This parameter is an INOUT parameter. It is used to correlate the events reported by multiple services for a unique received request. If the application has already received an event for that request through another manager interface, rApplicationData equals the value stored in it by the application. Otherwise, if it is the first event reported for this received request, rApplicationData is set to 0. The value of rApplicationData when EvAssertedIdentity returns is accessible through the GetOpaque() method of the ISipServerEventControl (see page 75) interface that accompanies the request when the owner service issues its event. This opaque data should be used to store the state indicating that the request contained P-Asserted-Identity header(s) and be able to act accordingly when the session manager receives the event containing the ISipServerEventControl (see page 75). Note that this event cannot be processed asynchronously since the opaque application data is passed by value to the ISipServerEventControl (see page 75) interface.

Description

This event is reported when a request is received from the trusted proxy server with valid P-Asserted-Identity header(s).

RFC 3325 states that a maximum of two headers can be present in a packet: one header with a SIP or SIPS URI and the other with a telephone URI. There may be one of the headers, both, or none. This event is not reported when there are none.

The identity contained in the P-Asserted-Identity header(s) may be considered "privileged, or intrinsically more trustworthy than the From header field of a request" (RFC 3325, section 8, first paragraph).

Also, RFC 3325 states that "If a UA is part of the Trust Domain from which it received a message containing a P-Asserted-Identity header field, then it can use the value freely but it MUST ensure that it does not forward the information to any element that is not part of the Trust Domain, if the user has requested that asserted identity information be kept private." A user requests for asserted identity information to be kept private by inserting the 'id' privacy type into the Privacy header. If the privacy type is present, the bPrivacyRequested parameter is true.

Finally, "If a UA is not part of the Trust Domain from which it received a message containing a P-Asserted-Identity header field, then it can assume this information does not need to be kept private."

3.7.1.38.1.2 - EvInvalidAssertedIdentity

3.7.1.38.1.2.1 - ISipUaAssertedIdentityMgr::EvInvalidAssertedIdentity Method

A response containing invalid P-Asserted-Identity has been received from the trusted proxy server.

C++

```
virtual void EvInvalidAssertedIdentity(IN ISipUaAssertedIdentitySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse, IN mxt_result resReason) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identities UA side on this context.
IN ISipClientEventControl* pClientEventCtrl	The client event control interface for this transaction. It cannot be NULL. The application must call either CallNextEvent or ClearClientEvents on it. If one of these methods is not called directly from the present method, a reference must be added on pClientEventControl. The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The response containing the invalid P-Asserted-Identity header(s).
IN mxt_result resReason	resFE_UNSUPPORTED_URI_SCHEME: A name-addr received in the P-Asserted-Identity header is not a valid name-addr for the header. It is neither a SIP, SIPS, or telephone URI
resFE_FAIL	There were too many of a URI type in the packet.

Description

This method is called when an invalid response is received from the trusted proxy server.

The response may be invalid because a P-Asserted-Identity header did not contain a valid URI (neither SIP, SIPS, or telephone),

because the response contained too many P-Asserted-Identity headers for the same URI type (for example, the response contained two P-Asserted-Identity headers with a telephone URI) or because the P-Asserted-Identity header is not applicable in a response to the associated request.

The application can ignore this event by calling pClientEventCtrl->CallNextClientEvent. Note that reissuing the request should be useless to fix this problem since it must come from a SIP entity between this UA and the server UA.

3.7.1.38.1.2.2 - ISipUaAssertedIdentityMgr::EvInvalidAssertedIdentity Method

A request containing invalid P-Asserted-Identity has been received from the trusted proxy server.

C++

```
virtual void EvInvalidAssertedIdentity(IN ISipUaAssertedIdentitySvc* pSvc, IN const CSipPacket& rRequest, IN
mxt_opaque opqApplicationData, IN mxt_result resReason) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identities UA side on this context.
IN const CSipPacket& rRequest	The request containing the invalid P-Asserted-Identity header(s).
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN mxt_result resReason	resFE_UNSUPPORTED_URI_SCHEME: A name-addr received in the P-Asserted-Identity header is not a valid name-addr for the header. It is neither a SIP, SIPS, or telephone URI
resFE_FAIL	There were too many of a URI type in the packet.

Description

This method is called when an invalid request is received.

The request may be invalid because a P-Asserted-Identity header did not contain a valid URI (neither SIP, SIPS, or telephone), because the request contained too many P-Asserted-Identity headers for the same URI type (for example, the request contained two P-Asserted-Identity headers with a SIP or SIPS URI) or because the usage of the P-Asserted-Identity header is not applicable for the request.

Before this event is reported, a 500 response is sent with the following reason phrase "Too many or erroneous P-Asserted-Identity header(s)".

3.7.1.38.1.3 - ISipUaAssertedIdentityMgr::EvTrustedProxyDnsResolutionCompleted Method New in 4.1.4

The DNS resolution of the trusted proxy URI list is completed.

C++

```
virtual void EvTrustedProxyDnsResolutionCompleted(IN ISipUaAssertedIdentitySvc* pSvc, IN bool
bSharedTrustedProxy) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identities UA side on this context.
IN bool bSharedTrustedProxy	<ul style="list-style-type: none"> • true: this event is for a SetSharedTrustedProxy call. • false: this event is for a SetInstanceTrustedProxy call.

Description

This method is called when the DNS resolution of the trusted proxy URI list is completed. It will be called for either the shared or instance trusted proxies as indicated by the bSharedTrustedProxy parameter.

Before this event is reported to the application, all other events of this manager SHOULD be considered unreliable. Once this event has been reported at least once for either the shared or instance trusted proxy list, all events are reliable.

This means that when the shared trusted proxies are set and the EvTrustedProxyDnsResolutionCompleted event has been reported

once, changing the instance trusted proxy will NOT render the events unreliable. Those events could be reported while the DNS resolution for that new list is in progress and they will be based on the shared list until the DNS resolution is completed.

The events reliability is based on the fact that there is at least one trusted proxy IP address list set. It does not matter if it is the shared or the instance one. As long as there is a trusted proxy IP address list set, the events are reliable according to that list.

The rule of thumb is: once the `EvTrustedProxyDnsResolutionCompleted` event has been reported, the service's events are reliable.

3.7.1.38.1.4 - `EvUntrustedProxy`

3.7.1.38.1.4.1 - `ISipUaAssertedIdentityMgr::EvUntrustedProxy` Method

A response has been received from an untrusted proxy.

C++

```
virtual void EvUntrustedProxy(IN ISipUaAssertedIdentitySvc* pSvc, IN ISipClientEventControl* pClientEventCtrl,
IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing Network Asserted Identity UA side on this context.
IN ISipClientEventControl* pClientEventCtrl	The client event control interface for this transaction. It cannot be NULL. The application must call either <code>CallNextEvent</code> or <code>ClearClientEvents</code> on it. If one of these methods is not called directly from the present method, a reference must be added on <code>pClientEventCtrl</code> . The added reference must be released after one of the methods is called.
IN const CSipPacket& rResponse	The response received from a proxy other than the trusted proxy.

Description

This event is reported when a response has been received from an untrusted entity.

The response may or may not contain one or more P-Asserted-Identity headers. If such headers are present, they must be ignored.

Before reporting this event, the service will have cleared any preferred identity previously configured. This is to prevent having the preferred identity information used in future requests and responses issued by a service on the associated SIP Context. The preferred identity information may be sensitive and is thus kept private. An application that still wants to present its preferred identity under this condition has to reconfigure its preferred identity.

The address of the entity from which the packet has been received can be retrieved by `rRequest.GetPeerAddr()`.

Note that this method should never be called when a request is sent directly to a trusted proxy.

3.7.1.38.1.4.2 - `ISipUaAssertedIdentityMgr::EvUntrustedProxy` Method

A request has been received from an untrusted proxy.

C++

```
virtual void EvUntrustedProxy(IN ISipUaAssertedIdentitySvc* pSvc, IN const CSipPacket& rRequest, INOUT
mxt_opaque& rApplicationData) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentitySvc* pSvc	The service managing the Network Asserted Identity UA side on this context.
IN const CSipPacket& rRequest	The request received from a proxy other than the trusted proxy.

INOUT mxt_opaque& rApplicationData	Application data opaque to the service. This parameter is an INOUT parameter. It is used to correlate the events reported by multiple services for a unique received request. If the application already received an event for that request through another manager interface, rApplicationData equals the value stored in it by the application. Otherwise, if it is the first event reported for this received request, rApplicationData is set to 0. The value of rApplicationData when EvUntrustedProxy returns is accessible through the GetOpaque() method of the ISipServerEventControl (see page 75) interface that accompanies the request when the owner service issues its event. This opaque data should be used to store the state indicating that the request contained has been received from a proxy it does not trust and be able to act accordingly when the session manager receives the event containing the ISipServerEventControl (see page 75). Note that this event cannot be processed asynchronously since the opaque application data is passed by value to the ISipServerEventControl (see page 75) interface.
------------------------------------	--

Description

This event is reported when a request has been received from an untrusted entity.

The request may or may not contain one or more P-Asserted-Identity headers. If such headers are present, they must be ignored.

Before reporting this event, the service will have cleared any preferred identity previously configured. This is to prevent having the preferred identity information used in future requests and responses issued by a service on the associated SIP Context. The preferred identity information may be sensitive and is thus kept private. An application that still wants to present its preferred identity under this condition has to reconfigure its preferred identity.

The address of the entity from which the packet has been received can be retrieved by calling rRequest.GetPeerAddr().

3.7.1.39 - ISipUaAssertedIdentitySvc Class

Class Hierarchy



C++

```
class ISipUaAssertedIdentitySvc : public IEComUnknown;
```

Description

The ISipUaAssertedIdentitySvc service implements Network Asserted Identity capabilities for a user agent.

This service allows the configuration of trusted proxies to which requests should be sent and from which they should be received. It will report an event when a request or a response is not received from a trusted proxy, in which case the application should ignore the P-Asserted-Identity header in the packet, if any. This service will also report an event when a request or a response is received from a trusted proxy with a P-Asserted-Identity header, in which case the application should use this asserted identity information instead of the From header of the request, or the To header of the response. Moreover, this service can be configured with a preferred identity, which will be placed into outgoing requests and responses.

It is the application's responsibility to ensure that the packet is sent to a trusted proxy when P-Preferred-Identity header(s) are added to the packet. Doing otherwise may compromise privacy. RFC 3325 states that "user agents MUST NOT populate the P-Preferred-Identity header field in a message that is not sent directly to a proxy that is trusted by the user agent."

To ensure that requests are sent to the trusted proxy server, the service user must configure a pre-loaded route, with the address of the trusted proxy as the first item in this pre-loaded route. If a response is received from another entity than one of the configured trusted proxies, the service will report the EvUntrustedProxy event, which means that the P-Asserted-Identity header that may be contained in the packet MUST NOT be used.

When an incoming request is not received from the trusted proxies, EvUntrustedProxy is reported to the manager. This event means that the service user must not consider the P-Asserted-Identity header found in this request.

The service allows the configuration of two identities (one SIP or SIPS URI and/or one telephone URI) to be added into packets as P-Preferred-Identity headers. These headers may be used by the trusted proxy to select the Network Asserted Identity to specify in the P-Asserted-Identity header it will generate. Note that nothing guarantees that the preferred identity will be used by the proxy. If the preferred identity is not known by the proxy, it can use its known Network Asserted Identity or it can reject the request with a 403 Forbidden.

When packets are received from a trusted proxy with the P-Asserted-Identity header, the service will warn the application through the ISipUaAssertedIdentityMgr (see page 623) interface. If the use of the P-Asserted-Identity is invalid (too many headers or invalid headers), the application will be notified by one of the EvInvalidAssertedIdentity events (depending on whether the packet is a request

or a response). Otherwise, one of the `EvAssertedIdentity` events will be reported.

To enable Network Asserted Identity privacy, use the `ISipPrivacySvc` (see page 507). This service allows to request for Network Asserted Identity privacy, which mandates the SIP elements in the Trust Domain to keep the user's information confidential.

This service notifies the application of events through the `ISipUaAssertedIdentityMgr` (see page 623) interface.

The `ISipUaAssertedIdentitySvc` is an ECOM object

The `ISipUaAssertedIdentitySvc` is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: `CLSID_CSipUaAssertedIdentitySvc`

Interface Id: `IID_ISipUaAssertedIdentitySvc`

A user can query the `ISipContext` (see page 23) to which this service is attached by calling `QueryIf` on it. It can also directly access all other services attached to the `ISipContext` (see page 23) through the same mean.

Location

`SipUserAgent/ISipUaAssertedIdentitySvc.h`

See Also

`ISipUaAssertedIdentityMgr` (see page 623), `ISipUserAgentSvc::SetPreloadedRoute` (see page 653)

Methods

Method	Description
•  <code>SetInstanceTrustedProxy</code> (see page 629)	Sets the address of the trusted proxy server that will be used by this instance of this service.
•  <code>SetManager</code> (see page 631)	Configures the manager associated with this service instance.
•  <code>SetPreferredIdentities</code> (see page 631)	Sets the identities added to the P-Preferred-Identity of the outgoing requests.
•  <code>SetSharedTrustedProxy</code> (see page 632)	Sets the address of the trusted proxy server that will be used by all instances of this service.

Legend

	Method
	abstract

3.7.1.39.1 - Methods

3.7.1.39.1.1 - `SetInstanceTrustedProxy`

3.7.1.39.1.1.1 - `ISipUaAssertedIdentitySvc::SetInstanceTrustedProxy` Method

Sets the address of the trusted proxy server that will be used by this instance of this service.

C++

```
virtual mxt_result SetInstanceTrustedProxy(IN const CSipUri& rTrustedProxy) = 0;
```

Parameters

Parameters	Description
IN const CSipUri& rTrustedProxy	The SIP / SIPS URI of the trusted proxy.

Returns

- `resFE_INVALID_STATE`: This service has no manager. Maybe it was not set or `Clear` was called on the context.

OR

The trusted proxy IP address list is currently used. The main cause for that would be that DNS resolution is currently processed for the service list.

- resS_OK: DNS resolution has begun for the trusted proxy URI passed to the method.

Description

Sets the address of the trusted proxy. It is through this proxy that the user's identity is asserted.

This method overrides, for this instance only, all previous and future configuration done through SetSharedTrustedProxy (see page 632). That is, this service will always use its instance configuration first, and if none is set, then it will use the shared configuration.

If the system only uses a single trusted proxy for all Sip Contexts, you should use SetSharedTrustedProxy (see page 632) once in the system as DNS resolution for the server will be done only once. If the system simultaneously uses more than one trusted proxy server, you will need to call SetInstanceTrustedProxy each time this service is attached to a Sip Context.

This method will resolve the SIP-URI / SIPS-URI to one or more IP addresses.

If a request is received from another proxy than the trusted proxy, the service will generate the EvUntrustedProxy event on its manager.

A pre-loaded route SHOULD be set by the application, with the first route pointing to the trusted proxy server. This is to guarantee that requests are always sent first to the trusted proxy server, which can then verify the user's identity before forwarding the request.

Notes

This method is kept for backward compatibility reasons. The overload that receives a

`CVector<CSipUri>`

parameter SHOULD be used instead.

See Also

`SetSharedTrustedProxy` (see page 632)

3.7.1.39.1.1.2 - **ISipUaAssertedIdentitySvc::SetInstanceTrustedProxy** Method New in 4.1.4

Sets the address of the trusted proxy servers that will be used by this instance of this service.

C++

```
virtual mxt_result SetInstanceTrustedProxy(IN const CVector<CSipUri>& rvecTrustedProxies) = 0;
```

Parameters

Parameters	Description
<code>IN const CVector<CSipUri>& rvecTrustedProxies</code>	The SIP / SIPS URI of the trusted proxies.

Returns

- `resFE_INVALID_STATE`: This service has no manager. Maybe it was not set or `Clear` was called on the context.

OR

The trusted proxy IP address list is currently used. The main cause for that would be that DNS resolution is currently processed for the service list.

- `resS_OK`: DNS resolution has begun for the list of trusted proxy URI passed to the method.

Description

Sets the address of the trusted proxy servers. It is through these proxies that the user's identity is asserted.

This method overrides, for this instance only, all previous and future configuration done through SetSharedTrustedProxy (see page 632). That is, this service will always use its instance configuration first, and if none is set, then it will use the shared configuration.

If the system only uses a single trusted proxy list for all Sip Contexts, you should use SetSharedTrustedProxy (see page 632) once in the system as DNS resolution for the server will be done only once. If the system simultaneously uses more than one trusted proxy server list, you will need to call SetInstanceTrustedProxy each time this service is attached to a Sip Context.

This method will resolve the SIP-URI / SIPS-URI list to one or more IP addresses for each URI.

If a request is received from another proxy than one of the trusted proxies, the service will generate the `EvUntrustedProxy` event on its manager.

A pre-loaded route SHOULD be set by the application, with the first route pointing to one of the trusted proxy server. This is to guarantee that requests are always sent first to a trusted proxy server, which can then verify the user's identity before forwarding the request.

See Also

SetSharedTrustedProxy (see page 632)

3.7.1.39.1.2 - **ISipUaAssertedIdentitySvc::SetManager** Method

Configures the manager associated with this service instance.

C++

```
virtual mxt_result SetManager(IN ISipUaAssertedIdentityMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipUaAssertedIdentityMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT: pMgr is NULL. The manager is not changed in this case.

resS_OK: Otherwise.

Description

Configures the manager that will receive the events reported by this service.

Note that a manager MUST be associated with this service before it is used.

3.7.1.39.1.3 - **ISipUaAssertedIdentitySvc::SetPreferredIdentities** Method

Sets the identities added to the P-Preferred-Identity of the outgoing requests.

C++

```
virtual mxt_result SetPreferredIdentities(IN const CNameAddr* pSipIdentity, IN const CNameAddr* pTelIdentity) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr* pSipIdentity	The SIP or SIPS URI identity to be added in a P-Preferred-Identity header. NULL if no SIP or SIPS URI identity should be sent in a P-Preferred-Identity header. By default, this parameter is NULL.
IN const CNameAddr* pTelIdentity	The telephone URI identity to be added in a P-Preferred-Identity header. NULL if no telephone URI identity should be sent in a P-Preferred-Identity header. By default, this parameter is NULL.

Returns

resFE_INVALID_STATE: The manager is NULL. This could happen when the manager was not set or when Clear was called on the ISipContext (see page 23).

resFE_INVALID_ARGUMENT: pSipIdentity is not NULL or does not contain a SIP or SIPS URI.

resFE_INVALID_ARGUMENT: pTelIdentity is not NULL or does not contain a telephone URI.

resS_OK: The configuration has been updated.

Description

Permits to configure the identities that the application prefers to be used as their Network Asserted Identity. These identities are added to P-Preferred-Identity headers.

To be RFC 3325 compliant, this method only permits to add a maximum of 2 P-Preferred-Identity headers: one with a SIP or SIPS URI and one with a telephone URI.

By default, no P-Preferred-Identity header is added to the outgoing packets.

Because this service must be used to manage P-Preferred-Identity headers, it removes all P-Preferred-Identity headers it finds in outgoing packets and then adds the one it creates following its configuration.

RFC 3325 states that a "user agents MUST NOT populate the P-Preferred-Identity header field in a message that is not sent directly to a proxy that is trusted by the user agent." It is the application's responsibility to properly route the requests directly to a trusted proxy.

To ask for privacy on the Network Asserted Identity, see `ISipPrivacySvc::SetPrivacyType` (see page 511).

3.7.1.39.1.4 - SetSharedTrustedProxy

3.7.1.39.1.4.1 - `ISipUaAssertedIdentitySvc::SetSharedTrustedProxy` Method

Sets the address of the trusted proxy server that will be used by all instances of this service.

C++

```
virtual mxt_result SetSharedTrustedProxy(IN const CSipUri& rTrustedProxy) = 0;
```

Parameters

Parameters	Description
IN const CSipUri& rTrustedProxy	The SIP / SIPS URI of the trusted proxy.

Returns

- `resFE_INVALID_STATE`: This service has no manager. Maybe it was not set or `Clear` was called on the context.

OR

This service already has its own instance list so modifying the shared list would not affect the current service behavior. Updating was not done.

OR

The trusted proxy IP address list is currently used. The main cause for that would be that DNS resolution is currently processed for the service list.

- `resS_OK`: DNS resolution has begun for the trusted proxy URI passed to the method.

Description

Sets the address of the trusted proxy. It is through this proxy that the user's identity is asserted.

This method configures all existing and future instances of this service with `rTrustedProxy`, but does not affect the instances that had their trusted proxy address configured through `SetInstanceTrustedProxy` (see page 629).

If the system only uses a single trusted proxy for all Sip Context, you should use `SetSharedTrustedProxy` once in the system as DNS resolution for the server will be done only once. If the system simultaneously uses more than one trusted proxy server, you will have to use `SetInstanceTrustedProxy` (see page 629) each time this service is attached to a Sip Context.

This method will resolve the SIP-URI / SIPS-URI to one or more IP addresses.

If a request is received from another proxy than the trusted proxy, the service will generate the `EvUntrustedProxy` event on its manager.

A pre-loaded route SHOULD be set by the application, with the first route pointing to the trusted proxy server. This is to guarantee that requests are always sent first to the trusted proxy server, which can then verify the user's identity before forwarding the request.

Notes

This method is kept for backward compatibility reasons. The overload that receives a

<code>CVVector<CSipUri></code>

parameter SHOULD be used instead.

See Also

`SetInstanceTrustedProxy` (see page 629)

3.7.1.39.1.4.2 - **ISipUaAssertedIdentitySvc::SetSharedTrustedProxy** Method New in 4.1.4

Sets the address of the trusted proxy servers that will be used by all instances of this service.

C++

```
virtual mxt_result SetSharedTrustedProxy(IN const CVector<CSipUri>& rvecTrustedProxies) = 0;
```

Parameters

Parameters	Description
IN const CVector<CSipUri>& rvecTrustedProxies	The SIP / SIPS URIs of the trusted proxies.

Returns

- resFE_INVALID_STATE: This service has no manager. Maybe it was not set or Clear was called on the context.

OR

This service already has its own instance list so modifying the shared list would not affect the current service behaviour. Updating was not done.

OR

The trusted proxy IP address list is currently used. The main cause for that would be that DNS resolution is currently processed for the service list.

- resS_OK: DNS resolution has begun for the list of trusted proxy URI passed to the method.

Description

Sets the address of the trusted proxy servers. It is through these proxies that the user's identity is asserted.

This method configures all existing and future instances of this service with rvecTrustedProxies, but does not affect the instances that had their trusted proxy addresses configured through SetInstanceTrustedProxy (see page 629).

If the system only uses a single trusted proxy list for all Sip Contexts, you should use SetSharedTrustedProxy once in the system as DNS resolution for the servers will be done only once. If the system simultaneously uses more than one trusted proxy server lists, you will need to use SetInstanceTrustedProxy (see page 629) each time this service is attached to a Sip Context.

This method will resolve the SIP-URI / SIPS-URI list to one or more IP addresses for each URI.

If a request is received from another proxy than one of the trusted proxies, the service will generate the EvUntrustedProxy event on its manager.

A pre-loaded route SHOULD be set by the application, with the first route pointing to one of the trusted proxy servers. This is to guarantee that requests are always sent first to a trusted proxy server, which can then verify the user's identity before forwarding the request.

Warning

The vector must not be empty or the shared trusted proxy list will be erased for all current and future instances.

See Also

[SetInstanceTrustedProxy](#) (see page 629)

3.7.1.40 - **ISipUpdateMgr** Class

Class Hierarchy

```
ISipUpdateMgr
```

C++

```
class ISipUpdateMgr;
```

Description

The update manager is the interface through which the update service talks to the application. It is through this interface that M5T SIP

UA informs the application of incoming UPDATE requests and of incoming responses to UPDATE requests sent to the remote party.

It is important to note that a special process is required by the application for the event `EvUpdated` (see page 636) when there is a pending offer to which no answer has been sent or received. Under some circumstances, the application has to answer a 491 or 500 response when there is a pending offer. Except for UPDATE request process, M5T SIP UA does not send such responses because it is the application's responsibility to look at the offers and to know whether or not there is a pending offer. M5T SIP UA automatically answers a 500 response to an UPDATE if it has received an incoming UPDATE to which no final answer has been sent. M5T SIP UA also automatically answers a 491 response to an UPDATE if it did not receive a final response to a previous outgoing UPDATE request. For more details on this, please see the annotated RFC 3311 Section 5.2 Conformance Items 12 to 19.

Upon reception of a 2xx response to a sent UPDATE request, the application MUST change the session description parameters according to what is found in the 2xx response. If a non-2xx response is received, the session description parameters MUST remain unchanged.

Location

`SipUserAgent/ISipUpdateMgr.h`

See Also

`ISipUpdateSvc` (see page 637), RFC 3311

Methods

Method	Description
• A EvFailure (see page 634)	Notifies the application that a final failure response has been received for the sent UPDATE request.
• A EvInvalidUpdate (see page 635)	Warns the application that an invalid UPDATE request packet has been received.
• A EvProgress (see page 635)	Notifies the application that a provisional response has been received for the sent UPDATE request.
• A EvSuccess (see page 636)	Notifies the application that a final successful response has been received for the sent UPDATE request.
• A EvUpdated (see page 636)	Notifies the application that an UPDATE request has been received.

Legend

• A	Method
A	abstract

3.7.1.40.1 - Methods

3.7.1.40.1.1 - ISipUpdateMgr::EvFailure Method

Notifies the application that a final failure response has been received for the sent UPDATE request.

C++

```
virtual void EvFailure(IN ISipUpdateSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response.

Description

Notifies the application that a final negative response has been received for the sent UPDATE request.

Depending on the status code of the response, different actions might be taken by the application.

In case the response is a 491, the application can start a timer with a randomly chosen value (See RFC 3311 section 3.2 for more information on how the value should be chosen). When the timer fires, the application can once more try to issue the UPDATE via `ISipClientEventControl::ReIssueRequest` (see page 20).

In case the response is 481 or 408, the application MUST terminate the dialog (by calling `ISipSessionSvc::Bye` (see page 580) for

example). Note that the M5T SIP stack reports a timeout by making a 408 response and reporting it to the application (through this method if the request sent was sent by the ISipUpdateSvc (see page 637)).

In all cases, the session description parameters MUST remain unchanged.

See Also

ISipUpdateSvc (see page 637)

3.7.1.40.1.2 - ISipUpdateMgr::EvInvalidUpdate Method

Warns the application that an invalid UPDATE request packet has been received.

C++

```
virtual void EvInvalidUpdate(IN ISipUpdateSvc* pSvc, IN mxt_opaque opqApplicationData, IN const CSipPacket& rRequest, IN mxt_result resReason) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateSvc* pSvc	The service managing this request.
IN mxt_opaque opqApplicationData	Data set by the application in another event for this same request. When this is the first event for this request, it is set to 0.
IN const CSipPacket& rRequest	The packet containing the invalid UPDATE request.
IN mxt_result resReason	resFE_INVALID_CONTACT_HEADER if the received UPDATE request did not contain a Contact header, contained an invalid Contact header, or contained more than one Contact header. In this case, an automatic 400 "Missing or Invalid Contact header field" response has been sent to the peer UAC. resFE_NO_ISIPUSERAGENTSVC_ATTACHED if an UPDATE request has been received while there was no ISipUserAgentSvc (see page 639) attached to the context. In this case, an automatic 500 response has been sent to the peer UAC. resFE_CALL_LEG_TRANSACTION_DOES_NOT_EXIST if an UPDATE request has been received while there was no dialog established. In this case, an automatic 481 response has been sent to the peer UAC. resFE_REQUEST_PENDING if an UPDATE request has been received while there was already an UPDATE request in progress. In this case, an automatic response has been sent to the peer UAC. If the UPDATE request in progress has been sent by the local UA, a 491 response is sent. If the request in progress has been sent by the peer UAC, a 500 response with a Retry-After header is sent.

Description

This method is called when an UPDATE request is received while the service cannot accept it.

Take note that if the manager of the service is not set, all the UPDATE requests are responded with a 500 response.

3.7.1.40.1.3 - ISipUpdateMgr::EvProgress Method

Notifies the application that a provisional response has been received for the sent UPDATE request.

C++

```
virtual void EvProgress(IN ISipUpdateSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response.

Description

Notifies the application that a provisional response has been received for the sent UPDATE request.

Upon reception of a provisional response, the session description parameters MUST remain unchanged.

See Also

[ISipUpdateSvc](#) (see page 637)

3.7.1.40.1.4 - ISipUpdateMgr::EvSuccess Method

Notifies the application that a final successful response has been received for the sent UPDATE request.

C++

```
virtual void EvSuccess(IN ISipUpdateSvc* pSvc, IN ISipClientEventControl* pClientEventCtrl, IN const CSipPacket& rResponse) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateSvc* pSvc	The service managing the response.
IN ISipClientEventControl* pClientEventCtrl	The interface to access client event related functionality.
rPacket	The response.

Description

Informs the application that a response has been received for the sent UPDATE request.

Upon reception of this event, the application MUST update the session description parameters to the value received in the packet.

Take note that the Request-Uri in the ISipUserAgentSvc (see page 639) is automatically updated to the value found in the Contact header of the response.

See Also

[ISipUpdateSvc](#) (see page 637)

3.7.1.40.1.5 - ISipUpdateMgr::EvUpdated Method

Notifies the application that an UPDATE request has been received.

C++

```
virtual void EvUpdated(IN ISipUpdateSvc* pSvc, IN ISipServerEventControl* pServerEventCtrl, IN const CSipPacket& rUpdateRequest) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateSvc* pSvc	The service managing the request.
IN ISipServerEventControl* pServerEventCtrl	The interface through which the application must return at least one response.
IN const CSipPacket& rUpdateRequest	The UPDATE request.

Description

Notifies the application that a remote user is trying to refresh the target by sending an UPDATE. The application should send responses through ISipServerEventControl::SendResponse (see page 76). The UPDATE MUST be promptly responded to. This means that the user MUST NOT be prompted to approve the session changes.

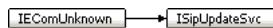
If the packet contains a payload, the application MUST inspect the offer contained within. If the application already has a pending generated or received offer (in an INVITE request or in a PRACK), it MUST reject the UPDATE with a 491 if the offer was generated by the local UA or with a 500 response if the offer was generated by the peer UA. The 500 response MUST have a Retry-After header with a random value between 0 and 10. If the application can generate an answer, it must put it in the 2xx response to the UPDATE. If the offer cannot be answered without prompting the user, the application SHOULD send a 504 response. If the offer is not acceptable, the application MUST answer with a 488 response. For more information on what response should be and what it must contain, refer to RFC 3311 in section 5.2.

Take note that if a 2xx response to the request is sent in response to this request, the Request-URI in the ISipUserAgentSvc (see page 639) is automatically updated to the value in the contact header of the request.

In some cases, an UPDATE request is automatically responded by the service. See EvInvalidUpdate (see page 635) for a list of these cases.

See Also

ISipUpdateSvc (see page 637), EvProgress (see page 635), EvSuccess (see page 636), EvFailure (see page 634), EvErroneousRequestPacket, RFC 3311

3.7.1.41 - ISipUpdateSvc Class**Class Hierarchy****C++**

```
class ISipUpdateSvc : public IEComUnknown;
```

Description

The update service implements RFC 3311. It is used to send an UPDATE request within a dialog (early or confirmed) to update session parameters without impacting the dialog state itself.

Under some circumstances, it is recommended not to send UPDATE requests containing an offer; for instance, when a pending offer is not answered yet. For more details on those circumstances, refer to RFC 3311 Section 5.1 (both caller and callee). The service sends an UPDATE request even if the application has a pending offer (other than UPDATE) for which no answer has been sent or received. It is the application's responsibility to manage the offer found in an UPDATE request when there is already a pending one.

The application must not send any UPDATE request while there is a pending incoming UPDATE request for which no final response has been provided yet. On the other hand, if M5T SIP UA receives an UPDATE request and it is waiting for the final response of an outgoing one, it automatically sends a 500 response. If it receives an UPDATE request while an UPDATE request was received but not answered yet, it automatically responds with a 491 response.

The application should add an Allow header containing the UPDATE request when sending an INVITE or a reliable provisional response to a pending INVITE. Doing so notifies the remote end that the local party supports the UPDATE mechanism.

Using the update service is done in collaboration with the session service and possibly the reliable response service (PRACK). PRACK can be used in conjunction with the UPDATE service to send offers in reliable provisional responses.

The application should only send UPDATE requests containing an offer when a pending INVITE is underway. Otherwise, it should use a re-INVITE instead. This is a requirement (conformance item 7) of RFC 3311 section 5.1.

This service generates events to the application through the ISipUpdateMgr (see page 633) interface.

The ISipUpdateSvc is an ECOM object

The ISipUpdateSvc is an ECOM object that is created and accessed through the following ECOM identifiers:

Class Id: CLSID_CSipUpdateSvc

Interface Id: IID_ISipUpdateSvc

A user can query the ISipContext (see page 23) to which this service is attached by calling QueryIf on it. It can also directly access all other services attached to the ISipContext (see page 23) through the same mean.

Location

SipUserAgent/ISipUpdateSvc.h

See Also

ISipSessionSvc::Invite (see page 581)

Methods

Method	Description
• A SetManager (see page 638)	Configures the manager associated with this service.
• A Update (see page 638)	Sends an UPDATE request.

Legend

	Method
	abstract

3.7.1.41.1 - Methods**3.7.1.41.1.1 - ISipUpdateSvc::SetManager Method**

Configures the manager associated with this service.

C++

```
virtual mxt_result SetManager(IN ISipUpdateMgr* pMgr) = 0;
```

Parameters

Parameters	Description
IN ISipUpdateMgr* pMgr	The manager. It must not be NULL.

Returns

resFE_INVALID_ARGUMENT if pMgr is NULL. The manager is not changed in this case.

resS_OK otherwise.

Description

Configures the manager that will receive the events generated by this service.

Note that a manager MUST be associated with this service before it is used. If no manager is associated with the service, the service automatically responds a 500 response to any received response.

3.7.1.41.1.2 - ISipUpdateSvc::Update Method

Sends an UPDATE request.

C++

```
virtual mxt_result Update(IN mxt_opaque opqTransaction, IN TO CHeaderList* pExtraHeaders, IN TO CSipMessageBody* pMessageBody, OUT ISipClientTransaction*& rpTransaction) = 0;
```

Parameters

Parameters	Description
IN mxt_opaque opqTransaction	Application data to associate with this transaction. This is opaque to the service.
IN TO CHeaderList* pExtraHeaders	Extra SIP headers to send with the request. Any Contact header in this list is replaced by the local contact of the ISipUserAgentSvc (see page 639) attached to the context. It can be NULL. Ownership is TAKEN.
IN TO CSipMessageBody* pMessageBody	Payload to include with the UPDATE. It can be NULL. Ownership is TAKEN.
OUT ISipClientTransaction*& rpTransaction	Interface to the created client transaction.

Returns

resFE_INVALID_STATE if there is no manager to this service. This could happen when no manager has been set on the service or when Clear was called on the context OR if there is no dialog OR if the local contact list is empty.

resFE_FAIL if there is no ISipUserAgentSvc (see page 639) attached to the context OR there is already an UPDATE request underway either UAC or UAS OR the request could not be sent OR the attached user agent service is incorrectly configured (e.g.: the contact list is empty).

resFE_SIPCORE_PACKET_BLOCKED: One of the service synchronously blocked the packet. No additional event will be reported.

resS_OK if the request has been sent.

Description

This method is used to send an UPDATE request to the remote target. Sending the request is possible only if the dialog is established between the UA and the peer. The application should only send UPDATE requests containing an offer when the pending original

INVITE is underway. Otherwise, it should use a re-INVITE instead.

While the INVITE request is not completed, an UPDATE request containing an offer should be sent only when a previous offer was generated (in the INVITE request) or received (in a reliable provisional response to the INVITE) and responded (in a reliable provisional response or in a PRACK). Refer to section 5.1 of RFC 3311 for more information on when an UPDATE request can be sent. This method sends an UPDATE request even if the application has a pending offer (other than UPDATE) for which no answer has been sent or received. It is the application's responsibility to manage the offer found in an UPDATE request when there is already a pending one.

The Contact header found in the ISipUserAgentSvc (see page 639) is automatically added to the UPDATE request.

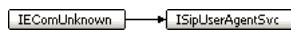
Notifications about the request's progress are received through the ISipUpdateMgr (see page 633) interface.

See Also

ISipUpdateMgr::EvProgress (see page 635), ISipUpdateMgr::EvSuccess (see page 636), ISipUpdateMgr::EvFailure (see page 634), ISipUserAgentSvc::GetState (see page 649), RFC 3311

3.7.1.42 - ISipUserAgentSvc Class

Class Hierarchy



C++

```
class ISipUserAgentSvc : public IEComUnknown;
```

Description

This interface is responsible to keep the dialog state for User Agent contexts. When using User Agent services like the session service or the generic service, the ISipUserAgentSvc MUST be added to the context except for cases provided below:

- The context is used to handle out-of-dialog or unsupported requests.
- When the same context is used to handle every blind NOTIFY requests related to MWI for instance.

This service is responsible for keeping dialog identifiers like From, To, and Call-ID headers and putting them in requests to send. This service also determines whether or not a packet belongs to this dialog. However, the dialog creation is dealt with automatically by the stack and the application does not have to bother with that(except when using the generic service).

When issuing a request, the application must configure this service with at least a target, the Contact (if the request to send can establish a dialog), the preloaded Route (unless the default empty Route is satisfying), and the From header. The service takes care of generating a unique Call-ID, a unique From Tag, a starting CSeq number, and an empty To tag. However, the application could force their value to something it specifies by calling the various configuration methods of this service. Note that once a request has been issued, the application can still change the dialog identifiers like the From Tag, To Tag, Call-ID, or CSeq number. However, it should refrain to do so as it could lead to unexpected behaviour like the remote peer not recognizing the request it receives or the local application not recognizing the requests received from the remote peer.

When the service is attached to a context that is created to receive a request, the dialog identifiers configure themselves when the dialog is established. The application only needs to configure a Contact (unless the application does not intend to create a dialog on that context).

Methods

Method	Description
• A AcceptOutOfOrderRemoteRequests (see page 640)	Sets the sequence number expected in the next request received.
• A AddLocalContact (see page 641)	Adds Contact header(s) in the internal Contact list.
• A CompleteDialogData (see page 641)	Completes the dialog information with the received packet.
• A Establish (see page 642)	Establishes a dialog with a packet sent or received.
• A EstablishForSentRequest (see page 643)	Establishes a dialog for a request to be sent.
• A GetCallId (see page 643)	Gets the Call-ID for this context.
• A GetCurrentTarget (see page 644)	Gets the current target where to send the requests.
• A GetInstancId (see page 644)	
• A GetLocalAddr (see page 644)	Gets the address to put in the From headers of outgoing requests.
• A GetLocalContact (see page 645)	Gets the Contact to put in Contact refresh requests and responses sent.
• A GetLocalContactVector (see page 645)	Gets the configured Contact list.

• A GetLocalDescriptorParameters (see page 645)	Gets the parameters to put in the From header of requests sent including the Tag parameter.
• A GetLocalSequenceNumber (see page 646)	Gets the sequence number to put in the requests to send.
• A GetLocalTag (see page 646)	Gets the tag parameter to put in the From header of requests sent.
• A GetPreloadedRoute (see page 646)	Gets the Route header to put in the out-of-dialog requests sent.
• A GetRemoteAddr (see page 647)	Gets the address to put in the To headers of outgoing requests.
• A GetRemoteDescriptorParameters (see page 647)	Gets the parameters to put in the To header of requests sent including the Tag parameter.
• A GetRemoteSequenceNumber (see page 648)	Gets the sequence number expected in the next request received.
• A GetRemoteSequenceNumber64bits (see page 648)	Gets the sequence number expected in the next request received.
• A GetRemoteTag (see page 649)	Gets the tag parameter to put in the To header of requests sent.
• A GetState (see page 649)	Obtains the dialog state.
• A SetCallId (see page 649)	Sets the Call-ID for this context.
• A SetCurrentTarget (see page 650)	Sets the current target where to send the requests.
• A SetInstanceld (see page 651)	
• A SetLocalAddr (see page 651)	Sets the address to put in the From headers of outgoing requests.
• A SetLocalDescriptorParameters (see page 652)	Sets the parameters to put in the From header of requests sent including the Tag parameter.
• A SetLocalSequenceNumber (see page 652)	Sets the sequence number to put in the requests to send.
• A SetLocalTag (see page 653)	Sets the tag parameter to put in the From header of requests sent.
• A SetPreloadedRoute (see page 653)	Set the Route header to put in the out-of-dialog requests sent.
• A SetRemoteAddr (see page 654)	Sets the address to put in the To headers of outgoing requests.
• A SetRemoteDescriptorParameters (see page 654)	Sets the parameters to put in the To header of requests sent including the Tag parameter.
• A SetRemoteSequenceNumber (see page 655)	Sets the sequence number expected in the next request received.
• A SetRemoteSequenceNumber64bits (see page 655)	Sets the sequence number expected in the next request received.
• A SetRemoteTag (see page 656)	Sets the tag parameter to put in the To header of requests sent.
• A TerminateUsage (see page 656)	Terminates a usage of a dialog.
• A UpdateRoute (see page 657)	Updates the Route set of a dialog.

Legend

•	Method
A	abstract

Enumerations

Enumeration	Description
EPacketDirection (see page 658)	
EState (see page 658)	
ETargetHeaderOption (see page 658)	This enum indicates if the target headers (of the actual current target that will be overwritten by the new current target) must be preserved and appended to the target headers of the new current target.

3.7.1.42.1 - Methods

3.7.1.42.1.1 - ISipUserAgentSvc::AcceptOutOfOrderRemoteRequests Method

Sets the sequence number expected in the next request received.

C++

```
virtual void AcceptOutOfOrderRemoteRequests(IN bool bAccept) = 0;
```

Parameters

Parameters	Description
IN bool bAccept	True to accept out of order remote requests or false to reject them.

Description

This method can be used to force the user agent service to accept remote requests regardless of the CSeq number set inside. In some

cases, a remote server could crash and try to properly keep a subscription alive but it will reset the CSeq number to a lower value.

By default the stack automatically rejects out of order requests.

NOTES: Setting this to true can lead to some unforeseen application behaviour and will render a UA non conformant to RFC 3261.

See Also

[SetRemoteSequenceNumber](#) (see page 655), [GetRemoteSequenceNumber](#) (see page 648)

3.7.1.42.1.2 - **ISipUserAgentSvc::AddLocalContact** Method

Adds Contact header(s) in the internal Contact list.

C++

```
virtual mxt_result AddLocalContact(IN TOS CSipHeader* pHdrContact) = 0;
```

Parameters

Parameters	Description
IN TOS CSipHeader* pHdrContact	The Contact header to add. If pHdrContact contains next headers, all headers will be inserted in the list. If pHdrContact contains a SIP-URI containing parameters other than "transport", their value may be overridden.

Returns

- **resFE_INVALID_ARGUMENT**: The header in parameter is not a valid Contact header.
- **resS_OK**: The Contact header has been successfully added.

Description

Adds a new Contact header in the local Contact header list. The Contact list is used by services to determine the Contact to put in the SIP requests and responses.

The Contact is selected by the SIP stack from the Contact list according to the local address used to send the SIP packet. Therefore, if the Contact is a FQDN, it must match one of the FQDNs added with the local addresses. If the Contact is an IP address, it must match one of the local IP addresses or address families.

If the application knows which local address the SIP stack uses to reach the peer assigned to the SIP context, it should add a single Contact, i.e., the Contact associated with the local address that is used. Otherwise, if the application never knows which local address is used to reach the peer, it should add a Contact for every local address configured in the SIP stack. Refer to [ISipCoreConfig::AddLocalAddress](#) (see page 33) for more details about local addresses.

It is important to understand that if pHdrContact contains raw data, this data is used without modification when creating SIP packets for the associated context. Refer to [CSipHeader](#) (see page 239) for more details on Raw versus Parsed header.

When the SIP-UA cannot find any configured contacts to match a network interface, the service will use the first contact that is available. This allows the application to easily override the automatic contact selection.

See Also

[GetLocalContact](#) (see page 645), [GetLocalContactVector](#) (see page 645)

3.7.1.42.1.3 - **ISipUserAgentSvc::CompleteDialogData** Method

Completes the dialog information with the received packet.

C++

```
virtual mxt_result CompleteDialogData(IN const CSipPacket& rReceivedPacket) = 0;
```

Returns

- **resFE_FAIL**: no dialog matcher list has been initialized.

- resFE_FAIL: the packet does not match the current dialog.
- resFE_FAIL: one or more mandatory headers are missing to get the complete dialog informations. These headers are Call-ID, From, To, and CSeq.
- resS_OK: the dialog information has been correctly initialized.

Description

This method MUST be called when a packet is received on the dialog initialized with EstablishForSentRequest (see page 643).

This method completes and updates the current dialog information into the ISipUserAgentSvc (see page 639).

See Also

EstablishForSentRequest (see page 643)

3.7.1.42.1.4 - ISipUserAgentSvc::Establish Method

Establishes a dialog with a packet sent or received.

C++

```
virtual mxt_result Establish(IN const CSipPacket& rPacket, IN EPacketDirection eDirection) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The packet that establishes the dialog.
IN EPacketDirection eDirection	eSENT when the packet is sent to a remote peer, eRECEIVED when the packet is received from a remote peer.

Returns

- resFE_FAIL: A dialog has already been established on this service

AND the dialog identifiers of the packet in parameter do not match those of this dialog. Multiple calls to establish happen for instance when doing call transfer with REFER on an existing INVITE dialog. In this case, the dialog identifiers of the 2xx to the REFER or those of the NOTIFY MUST match the dialog identifiers of the INVITE dialog. If they do not match, the REFER or the NOTIFY do not belong to that dialog in the first place and must be handled by another context.

This value is also returned for an attempt to establish a dialog with a '100 Trying' or a '>=300' response.

- resSW_DIALOG_NOT_REGISTERED: All the processing was done to establish the dialog but the dialog was not registered in the dialog matcher list. This happens only when Clear was called on the context, so the service will be destroyed soon. It means that no request matching the dialog will be received.
- resS_OK: The dialog is properly initialized.

Description

This method MUST be called when a packet that establishes a dialog is sent or received. Normally, it is called by the service that handles this type of dialog (EventSvc, SessionSvc).

If the application implements a new type of request that establishes a dialog through the GenericSvc, it MUST call this method itself. The GenericSvc never calls this method when used as a UAC. It calls this method as a UAS when its ISipDialogServerEventCtrl interface is used to send a response for a received request.

Once this method is called, this service registers itself with the CSipDialogMatcherList if it is not already done. It also switches to a state where it only accepts requests and responses for which the dialog identifiers (local tag, remote tag, Call-ID) match those of the packet in parameter. It also automatically manages out-of-order CSeq number in received requests on this dialog.

Note that this service has no knowledge of when the dialog terminates. The entity that called this method is also responsible to call

TerminateUsage (see page 656) in order to terminate the dialog.

In the particular case of the GenericSvc, it is the application's responsibility to terminate the dialog, even when the service itself called this method. The call was made on behalf of the application and it is considered as if the application made it itself.

See Also

TerminateUsage (see page 656), EstablishForSentRequest (see page 643)

3.7.1.42.1.5 - ISipUserAgentSvc::EstablishForSentRequest Method

Establishes a dialog for a request to be sent.

C++

```
virtual mxt_result EstablishForSentRequest() = 0;
```

Returns

- resFE_FAIL: no dialog matcher list has been initialized.

- resSW_DIALOG_NOT_REGISTERED: All the processing was done to

establish the dialog but the dialog was not registered in the dialog matcher list. This happens only when Clear was called on the context, so the service will be destroyed soon. It means that no request matching the dialog will be received anymore.

- resS_OK: The dialog is properly created.

Description

This method MUST be called when a request to be sent needs to establish a dialog. This method should only be called when requests should be accepted between the time the request is sent and the time its final response is received (SUBSCRIBE for example). This method should only be used in the services managing this type of method. If the application implements a new type of request that establishes a dialog through the GenericSvc, it MUST call this method itself. The GenericSvc never calls this method.

Once this method is called, this service registers itself with the CSipDialogMatcherList if it is not already done. It also switches to a state where it only accepts requests and responses for which the dialog identifiers (local tag, remote tag, Call-ID) match those configured in the service. Note that every request or response with the same local tag and Call-ID matches this dialog until one is received. When so, only requests and responses with the same dialog identifiers are accepted. This service also automatically manages out-of-order CSeq number in received requests on this dialog.

Note that this service has no knowledge of when the dialog terminates. The entity that called this method is also responsible to call TerminateUsage (see page 656) in order to terminate the dialog.

The CompleteDialogData (see page 641) method must be used when all dialog informations are known. This means when the first of a request received for the same dialog or a non-100 response to the request that was sent.

See Also

TerminateUsage (see page 656), Establish (see page 642)

3.7.1.42.1.6 - ISipUserAgentSvc::GetCallId Method

Gets the Call-ID for this context.

C++

```
virtual const CSipHeader& GetCallId() const = 0;
```

Returns

The Call-ID to put in outgoing requests and that must be found in incoming requests.

Description

This parameter has two usages. The first usage is the Call-ID header of requests sent through the context to which this user agent service is attached. It is put as the Call-ID header of requests created with CreateBasicRequest.

The second usage is to match the Call-ID of incoming packets when this dialog is in the eIN_DIALOG state.

By default, the stack generates a new random Call-ID for each new user agent service instance. It is also automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

The Call-ID can also be set through SetCallId (see page 649).

See Also

CreateBasicRequest, CSipDialogMatcherList, SetCallId (see page 649)

3.7.1.42.1.7 - ISipUserAgentSvc::GetCurrentTarget Method

Gets the current target where to send the requests.

C++

```
virtual const IUri* GetCurrentTarget() const = 0;
```

Returns

The URI to put in the Request-URI of outgoing requests. It can be NULL.

Description

Gets the URI used as the Request-URI when creating a request with CreateBasicRequest. It returns NULL if the current target is uninitialized.

It can be set through SetCurrentTarget (see page 650).

See Also

CreateBasicRequest, SetCurrentTarget (see page 650)

3.7.1.42.1.8 - ISipUserAgentSvc::GetInstanceId Method New in 4.1.6

```
virtual const CString& GetInstanceId() const = 0;
```

Returns

The instance-id.

Description

Gets the instance-id of this user-agent.

3.7.1.42.1.9 - ISipUserAgentSvc::GetLocalAddr Method

Gets the address to put in the From headers of outgoing requests.

C++

```
virtual const CNameAddr& GetLocalAddr() const = 0;
```

Returns

The named-addr to put in the From headers of outgoing requests. It can be invalid.

Description

Gets the name addr to put in the From header of requests created with CreateBasicRequest.

This parameter is not used in dialog matching.

Note that this parameter is not directly used as the named-addr in the To header of the response sent. The To header of the received request is rather copied into the responses.

When a context is established, the information in the packet in parameter to the Establish (see page 642) method is used. This parameter must be valid for CreateBasicRequest to succeed.

This parameter can be set through SetLocalAddr (see page 651).

See Also

[SetLocalAddr](#) (see page 651), [CreateBasicRequest](#), [Establish](#) (see page 642)

3.7.1.42.1.10 - ISipUserAgentSvc::GetLocalContact Method

Gets the Contact to put in Contact refresh requests and responses sent.

C++

```
virtual const CSipHeader& GetLocalContact() const = 0;
```

Returns

The Contact header to put in Contact refresh requests and responses.

Description

Gets the Contact used by services to determine the Contact to put in the Contact refresh requests and responses to those requests. This method is usually used by services that send or receive target refresh requests or responses, but the application must set this parameter value through [AddLocalContact](#) (see page 641).

See Also

[AddLocalContact](#) (see page 641), [GetLocalContactVector](#) (see page 645)

3.7.1.42.1.11 - ISipUserAgentSvc::GetLocalContactVector Method

Gets the configured Contact list.

C++

```
virtual CVector<CSipHeader*>& GetLocalContactVector() = 0;
```

Returns

The vector of the Contact header.

Description

Gets the Contact vector used by services to determine the Contact to put in the Contact refresh requests and responses to those requests. This method should be used by the application to change a Contact, however Contact addition should be made through the [AddLocalContact](#) (see page 641) method.

See Also

[AddLocalContact](#) (see page 641), [GetLocalContact](#) (see page 645)

3.7.1.42.1.12 - ISipUserAgentSvc::GetLocalDescriptorParameters Method

Gets the parameters to put in the From header of requests sent including the Tag parameter.

C++

```
virtual const CGenParamList* GetLocalDescriptorParameters() const = 0;
```

Returns

A pointer to the list of parameters to put in the From header of requests to send. It can be NULL, which means that no parameters are put in the From header.

Description

Gets the parameters to put in the From header of requests created with [CreateBasicRequest](#).

These parameters include the Tag parameter. The local Tag parameter usage is described in [GetLocalTag](#) (see page 646).

This parameter list does not need to be configured by the application. It is automatically updated with the information in the parameter to [Establish](#) (see page 642) when this method is called.

This parameter list can be set through [SetLocalDescriptorParameters](#) (see page 652).

See Also

[GetLocalTag](#) (see page 646), [SetLocalDescriptorParameters](#) (see page 652)

3.7.1.42.1.13 - ISipUserAgentSvc::GetLocalSequenceNumber Method

Gets the sequence number to put in the requests to send.

C++

```
virtual uint32_t GetLocalSequenceNumber() const = 0;
```

Returns

uSequenceNumber: The sequence number to put in the requests to send.

Description

Gets the sequence number put in the CSq header of the next requests created with CreateBasicRequest.

This parameter is not used locally to match the dialog.

This parameter is increased by one each time CreateBasicRequest is called successfully.

By default, the user agent service initializes this value with a random number between 1 and (2³¹ - 1) inclusively.

This value can be set through SetLocalSequenceNumber (see page 652).

See Also

[SetLocalSequenceNumber](#) (see page 652), [CreateBasicRequest](#)

3.7.1.42.1.14 - ISipUserAgentSvc::GetLocalTag Method

Gets the tag parameter to put in the From header of requests sent.

C++

```
virtual const CString& GetLocalTag() const = 0;
```

Returns

The tag parameter to put in the From header of requests sent. It can be an empty string, which means that no tag is put in the From header.

Description

Gets the tag parameter to put in the From header of requests created with CreateBasicRequest.

This parameter is also used to match the dialog for incoming packets when the user agent service is in the eIN_DIALOG state. When in such a state, its value must match the tag of the From header of incoming responses or the tag of the To header of incoming requests. If it does not match, the dialog does not match and the packet must be handled by another context.

This parameter does not need to be configured by the application. A new random value is generated for every new user agent service instance. The value is also updated from the packet in parameter when Establish (see page 642) is called.

This parameter can be set through SetLocalTag (see page 653).

It is contained in the parameters returned by GetLocalDescriptorParameters (see page 645) unless it is an empty string.

See Also

[SetLocalTag](#) (see page 653), [GetLocalDescriptorParameters](#) (see page 645), [CreateBasicRequest](#), [Establish](#) (see page 642).

3.7.1.42.1.15 - ISipUserAgentSvc::GetPreloadedRoute Method

Gets the Route header to put in the out-of-dialog requests sent.

C++

```
virtual const CSipHeader* GetPreloadedRoute() const = 0;
```

Returns

The Route headers to put in out-of-dialog client requests. It can be NULL, which means that no preloaded Route is used.

Description

Gets the Route headers to put in the out-of-dialog requests sent by contexts that use this ISipUserAgentSvc (see page 639).

This Route is used only when no dialog is established yet (such as when sending a dialog establishing INVITE or sending a REGISTER). When a dialog is established, the rules of RFC 3261 dictate how the Route is computed according to the requests and responses that established the dialog.

See Also

GetPreloadedRoute

3.7.1.42.1.16 - ISipUserAgentSvc::GetRemoteAddr Method

Gets the address to put in the To headers of outgoing requests.

C++

```
virtual const CNameAddr& GetRemoteAddr() const = 0;
```

Returns

The named-addr to put in the To headers of outgoing requests. It can be invalid.

Description

Gets the name addr to put in the To header of requests created with CreateBasicRequest.

This parameter is not used in dialog matching.

Note that this parameter is not directly used as the named-addr in the From header of the response sent. The From header of the received request is rather copied into responses.

This parameter does not have a default value and must be set when the user agent service is attached to a context created to send a request. It is not necessary to set its value if the context is used only to receive requests until a dialog (early or not) is established on it (through the Establish (see page 642) method). When Establish (see page 642) is called, this parameter is overridden with the information in the packet in parameter. To send an out-of-dialog request (including one that can establish a new dialog), the application must set this parameter through SetRemoteAddr (see page 654).

This method returns an invalid CNameAddr (see page 201) if this parameter is uninitialized.

See Also

SetRemoteAddr (see page 654), CreateBasicRequest, Establish (see page 642)

3.7.1.42.1.17 - ISipUserAgentSvc::GetRemoteDescriptorParameters Method

Gets the parameters to put in the To header of requests sent including the Tag parameter.

C++

```
virtual const CGenParamList* GetRemoteDescriptorParameters() const = 0;
```

Returns

A pointer to the list of parameters to put in the To header of requests to send. It can be NULL, which means that no parameter would be added in the To header.

Description

Gets the header parameters to put in the To header of requests created with CreateBasicRequest.

These parameters include the Tag parameter. The remote Tag parameter usage is described in GetRemoteTag (see page 649).

This parameter list does not need to be configured by the application. It is automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

This parameter list can be set through SetRemoteDescriptorParameters (see page 654).

See Also

[SetRemoteTag](#) (see page 656), [SetRemoteDescriptorParameters](#) (see page 654)

3.7.1.42.1.18 - ISipUserAgentSvc::GetRemoteSequenceNumber Method

Gets the sequence number expected in the next request received.

C++

```
virtual uint32_t GetRemoteSequenceNumber() const = 0;
```

Returns

The expected remote sequence number.

Description

Gets the sequence number that is expected in the CSeq of the next request received on this dialog.

This parameter has meaning only when the state is eIN_DIALOG. When the state is eOUT_OF_DIALOG, it has absolutely no effect. When the state is eIN_DIALOG, the user agent service automatically rejects requests for which the CSeq is smaller than or equal to this parameter.

The application does not need to set this parameter. By default it is initialized with the value in the CSeq header of the packet in parameter to Establish (see page 642) when this method is called with a received request or a sent response. Otherwise, it is initialized with the CSeq value of the first request received when eIN_DIALOG is true.

If one changes its value after it has been initialized, it could lead the application to either accept out-of-order requests from the remote peer or reject its valid requests as out-of-order requests.

This value can be set through [SetRemoteSequenceNumber](#) (see page 655).

See Also

[Establish](#) (see page 642), [SetRemoteSequenceNumber](#) (see page 655)

3.7.1.42.1.19 - ISipUserAgentSvc::GetRemoteSequenceNumber64bits Method

Gets the sequence number expected in the next request received.

C++

```
virtual uint64_t GetRemoteSequenceNumber64bits() const = 0;
```

Returns

The expected remote sequence number.

Description

Gets the sequence number that is expected in the CSeq of the next request received on this dialog.

This parameter has meaning only when the state is eIN_DIALOG. When the state is eOUT_OF_DIALOG it has absolutely no effect. When the state is eIN_DIALOG, the user agent service automatically rejects requests for which the CSeq is smaller than or equal to this parameter.

The application does not need to set this parameter. By default it is initialized with the value in the CSeq header of the packet in parameter to Establish (see page 642) when this method is called with a received request or a sent response. Otherwise, it is initialized with the CSeq value of the first request received when eIN_DIALOG is true.

If one changes its value after it has been initialized, it could lead the application to either accept out of order requests from the remote peer or reject its valid requests as out of order requests.

This value can be set through [SetRemoteSequenceNumber](#) (see page 655).

See Also

[Establish](#) (see page 642), [SetRemoteSequenceNumber](#) (see page 655)

3.7.1.42.1.20 - ISipUserAgentSvc::GetRemoteTag Method

Gets the tag parameter to put in the To header of requests sent.

C++

```
virtual const CString& GetRemoteTag() const = 0;
```

Returns

The tag parameter to put in the To header of requests sent. It can be an empty string, which means that no tag is put in requests sent.

Description

Gets the tag parameter to put in the To header of requests created with CreateBasicRequest.

This parameter is also used to match the dialog for incoming packets when the user agent service is in the eIN_DIALOG state. When in such state, its value must match the tag of the To header of incoming responses or the tag of the From header of incoming requests. If it does not match, the dialog does not match and the packet must be handled by another context.

The exception to this matching rule is when a forking request is sent and a forked packet is received. In this case, and when the dialog state is in eOUT_OF_DIALOG, the remote tag parameter is not used to match the dialog.

This parameter does not need to be configured by the application. It is automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

This parameter can be set through SetRemoteTag (see page 656).

It is contained in the parameters returned by GetRemoteDescriptorParameters (see page 647) unless it is an empty string.

See Also

SetRemoteTag (see page 656), GetRemoteDescriptorParameters (see page 647), CreateBasicRequest, Establish (see page 642).

3.7.1.42.1.21 - ISipUserAgentSvc::GetState Method

Obtains the dialog state.

C++

```
virtual EState GetState() const = 0;
```

Returns

- eIN_DIALOG: There is at least one active dialog usage on this context.
- eWAITING_FOR_DIALOG_COMPLETION: Sent a request that permits the peer to create the dialog with a request. For example, sent a SUBSCRIBE, for which a NOTIFY can be received before a response.
- eOUT_OF_DIALOG: There is no dialog usage on this context.

Description

Returns the dialog current state.

See Also

Establish (see page 642), TerminateUsage (see page 656)

3.7.1.42.1.22 - ISipUserAgentSvc::SetCallId Method

Sets the Call-ID for this context.

C++

```
virtual mxt_result SetCallId(IN const CSipHeader& rCallID) = 0;
```

Parameters

Parameters	Description
IN const CSipHeader& rCallID	The Call-ID to put in outgoing requests and that must be found in incoming requests.

Returns

- resFE_INVALID_ARGUMENT: The header in parameter is not a valid Call-ID header.
- resSW_DIALOG_NOT_REGISTERED: All the processing was done to change the Call-ID, but the dialog was already established and Clear was called on the context so the new Call-ID was not registered in the dialog matcher list. It means that no request matching the dialog will be received. Note that the previous Call-ID was unregistered from the dialog matcher list when Clear was called, so no further information will be received from it either.
- resS_OK: The Call-ID has been successfully updated.

Description

This parameter has two usages. The first usage is the Call-ID header of requests sent through the context to which this user agent service is attached. It is used as the Call-ID header of requests created with CreateBasicRequest.

The second usage is to match the Call-ID of incoming packets when this dialog is in the eIN_DIALOG state.

One should not change this parameter after a dialog has been established on this user agent service as it has the effect of breaking the dialog matching mechanism both at the local application and the remote peer.

This parameter does not need to be configured by the application. By default, the stack generates a new random Call-ID for each new user agent service instance. It is also automatically updated with the information in the parameter to Establish (see page 642) when this method is called. The default Call-ID can still be accessed through the GetCallId (see page 643) method.

It is important to understand that if rCallID contains raw data, this data is used without modification when creating SIP packets for the associated context. Refer to CSipHeader (see page 239) for more details on Raw versus Parsed header.

See Also

GetCallId (see page 643), CreateBasicRequest, CSipDialogMatcherList

3.7.1.42.1.23 - ISipUserAgentSvc::SetCurrentTarget Method

Sets the current target where to send the requests.

C++

```
virtual mxt_result SetCurrentTarget(IN const IUri* pUri, IN ETARGETHEADEROPTION eTargetHeaderOption = eCLEAR_TARGET_HEADERS) = 0;
```

Parameters

Parameters	Description
IN const IUri* pUri	The URI to put in Request-URI of outgoing requests. It must not be NULL.
IN ETARGETHEADEROPTION eTargetHeaderOption = eCLEAR_TARGET_HEADERS	Whether the headers of the actual current target are appended to the headers of the new current target or not.

Returns

- resFE_INVALID_ARGUMENT: The URI in parameter is NULL.
- resFE_INVALID_ARGUMENT: eTargetHeaderOption is equal to ePRESERVE_TARGET_HEADERS and the current target contains headers but pUri is not a SIP or SIPS URI so the headers cannot be copied.

- resS_OK: The target has been successfully updated.

Description

This method should be called in two distinct occasions. The first occasion is when creating a context to send requests. The application would create the context and attach the appropriate services including the ISipUserAgentSvc (see page 639) and it would then call this method with a valid URI to use in the Request-URI of requests to send.

The second occasion where this method is called is when a target refresh request or response is received. Services that handle such requests are responsible to call that method themselves and the application does not need to bother with that. The only exception is when the application handles target refresh request or responses through the ISipGenericSvc (see page 481). In this case, it must call this method itself with the URI found in the Contact of the request or the response received.

The headers of the current target are used in the next request to the specified target. As long as they are not changed, these headers are used in every request sent to the specified target. When calling this method, these headers can be preserved and appended to the headers of pUri (by specifying ePRESERVE_TARGET_HEADERS). The headers cannot be preserved if the type of the pUri is not SIP or SIPS.

The URI set in this method is used as the request-URI when creating a request with CreateBasicRequest. It can be consulted through GetCurrentTarget (see page 644).

See Also

GetCurrentTarget (see page 644), CreateBasicRequest

3.7.1.42.1.24 - ISipUserAgentSvc::SetInstanceId Method New in 4.1.6

```
virtual void SetInstanceId(IN const CString& rstrInstanceId) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrInstanceId	The instance-id.

Description

Sets the instance-id used to add in Contacts of REGISTER requests sent over an outbound connection. This instance-id is unique to each user-agent.

See Also

ISipOutboundConnectionSvc (see page 102)

3.7.1.42.1.25 - ISipUserAgentSvc::SetLocalAddr Method

Sets the address to put in the From headers of outgoing requests.

C++

```
virtual void SetLocalAddr(IN const CNameAddr& rNameAddr) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rNameAddr	The named-addr to put in the From headers of outgoing requests.

Description

Sets the name addr to put in the From headers of requests created with CreateBasicRequest.

This parameter is not used in dialog matching. However, the application should be aware that changing this parameter while the user agent service is in the eIN_DIALOG state can cause interoperability failure with older user agent that comply with RFC 2543 instead of RFC 3261. This is because this parameter was used in dialog matching then.

Note that this parameter is not directly used as the named-addr in the To header of the response sent. Instead, the To header of the received request is copied into responses.

When a context is established, the information in the packet in parameter to the Establish (see page 642) method is used. This

parameter must be valid for CreateBasicRequest to succeed.

This parameter can be obtained through GetLocalAddr (see page 644).

See Also

GetLocalAddr (see page 644), CreateBasicRequest, Establish (see page 642)

3.7.1.42.1.26 - ISipUserAgentSvc::SetLocalDescriptorParameters Method

Sets the parameters to put in the From header of requests sent including the Tag parameter.

C++

```
virtual mxt_result SetLocalDescriptorParameters(IN TO CGenParamList* pParameters) = 0;
```

Parameters

Parameters	Description
IN TO CGenParamList* pParameters	The list of parameters to put in the From header of requests to send. Ownership is taken unless an error is returned.

Returns

- resFE_INVALID_ARGUMENT: The parameter list contains invalid parameters. Ownership of the parameter list is not taken.
- resS_OK: The parameter list has been successfully updated.

Description

Sets the header parameters to put in the From header of requests created with CreateBasicRequest.

These parameters include the Tag parameter. The local Tag parameter usage is described in SetLocalTag (see page 653).

One should take the same care not to change the local tag with this method once a dialog is established as with the SetLocalTag (see page 653) method. Other parameters can be changed but be aware that changing such parameters could lead to interoperability problems with non compliant peers.

This parameter list does not need to be configured by the application. It is automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

This parameter list can be obtained through GetLocalDescriptorParameters (see page 645).

Note that these parameters are not used to create the To header parameters of the responses sent. The To header of the responses sent is copied from the request received, adding a Tag parameter if none is present.

See Also

SetLocalTag (see page 653), GetLocalDescriptorParameters (see page 645)

3.7.1.42.1.27 - ISipUserAgentSvc::SetLocalSequenceNumber Method

Sets the sequence number to put in the requests to send.

C++

```
virtual void SetLocalSequenceNumber(IN uint32_t uSequenceNumber) = 0;
```

Parameters

Parameters	Description
IN uint32_t uSequenceNumber	The sequence number to put in the requests to send.

Description

Sets the sequence number put in the CSeq header of the requests created with CreateBasicRequest.

This parameter is not used locally to match the dialog. However, one should take care when changing this parameter. If its value is

lowered after a request has been sent, the remote peer will probably consider the requests it receives out of order and answer them with a 500 failure response.

This parameter is increased by one each time CreateBasicRequest is called successfully.

By default, the user agent service initializes this value with a random number between 1 and (2^31 - 1) inclusively.

This value is accessible through GetLocalSequenceNumber (see page 646).

See Also

GetLocalSequenceNumber (see page 646), CreateBasicRequest

3.7.1.42.1.28 - ISipUserAgentSvc::SetLocalTag Method

Sets the tag parameter to put in the From header of requests sent.

C++

```
virtual void SetLocalTag(IN const CString& rstrTag) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrTag	The tag parameter to put in the From header of requests sent.

Description

Sets the tag parameter to put in the From header of requests created with CreateBasicRequest.

This parameter is also used to match the dialog for incoming packets when the user agent service is in the eIN_DIALOG state. When in such a state, its value must match the tag of the From header of incoming responses or the tag of the To header of incoming requests. If it does not match, the dialog does not match and the packet must be handled by another context.

One should not change this parameter after a dialog has been established on this user agent service as it has the effect of breaking the dialog matching mechanism both at the local application and the remote peer.

This parameter does not need to be configured by the application. A new random value is generated for every new user agent service instance. The value is also updated from the packet in parameter when Establish (see page 642) is called.

This parameter can be obtained through GetLocalTag (see page 646). It is also contained in the parameters returned by GetLocalDescriptorParameters (see page 645) unless it is an empty string.

See Also

GetLocalTag (see page 646), GetLocalDescriptorParameters (see page 645), CreateBasicRequest, Establish (see page 642).

3.7.1.42.1.29 - ISipUserAgentSvc::SetPreloadedRoute Method

Set the Route header to put in the out-of-dialog requests sent.

C++

```
virtual mxt_result SetPreloadedRoute(IN TO CSipHeader* pPreloadedRoute) = 0;
```

Parameters

Parameters	Description
IN TO CSipHeader* pPreloadedRoute	The Route headers to put in out-of-dialog client requests. It can be NULL, which means that no preloaded Route is used. The ownership of this parameter is taken by this method unless an error is returned.

Returns

- resFE_INVALID_ARGUMENT: When the pPreloadedRoute is not NULL and at least one of the headers it contains is not a Route header. Ownership of the parameter is not taken in this case.
- resS_OK: The preloaded route has been successfully updated.

Ownership of the parameter is taken.

Description

Sets the Route headers to put in out-of-dialog requests sent by contexts that use this ISipUserAgentSvc (see page 639).

This Route is used only when no dialog is established yet (such as when sending a dialog establishing INVITE or sending a REGISTER). When a dialog is established, the rules of RFC 3261 dictate how the Route is computed according to requests and responses that established the dialog.

It is important to understand that if pPreloadedRoute contains raw data, this data is used without modification when creating SIP packets for the associated context. Refer to CSipHeader (see page 239) for more details on Raw versus Parsed header.

See Also

GetPreloadedRoute (see page 646)

3.7.1.42.1.30 - ISipUserAgentSvc::SetRemoteAddr Method

Sets the address to put in the To headers of outgoing requests.

C++

```
virtual void SetRemoteAddr(IN const CNameAddr& rNameAddr) = 0;
```

Parameters

Parameters	Description
IN const CNameAddr& rNameAddr	The named-addr to put in the To headers of outgoing requests.

Description

Sets the name addr to put in the To header of requests created with CreateBasicRequest.

This parameter is not used in dialog matching. However, the application should be aware that changing this parameter while the user agent service is in the eIN_DIALOG state can cause interoperability failure with older user agent that comply with RFC 2543 instead of RFC 3261. This is because this parameter was used in dialog matching then.

Note that this parameter is not directly used as the named-addr in the From header of the response sent. Instead the From header of the received request is copied into responses.

This parameter does not have a default value and must be set when the user agent service is attached to a context created to send a request. It is not necessary to set its value if the context is used only to receive requests until a dialog (early or not) is established on it (through the Establish (see page 642) method). When Establish (see page 642) is called, this parameter is overridden with the information in the packet in parameter.

This parameter can be obtained through GetRemoteAddr (see page 647).

See Also

GetRemoteAddr (see page 647), CreateBasicRequest, Establish (see page 642)

3.7.1.42.1.31 - ISipUserAgentSvc::SetRemoteDescriptorParameters Method

Sets the parameters to put in the To header of requests sent including the Tag parameter.

C++

```
virtual mxt_result SetRemoteDescriptorParameters(IN TO CGenParamList* pParameters) = 0;
```

Parameters

Parameters	Description
IN TO CGenParamList* pParameters	The list of parameters to put in the To header of requests to send. Ownership is taken unless an error is returned.

Returns

- resFE_INVALID_ARGUMENT: The parameter list contains invalid parameters. Ownership of the parameter list is not taken.

- resS_OK: The parameter list has been successfully updated.

Description

Sets the header parameters to put in the To header of requests created with CreateBasicRequest.

These parameters include the Tag parameter. The remote Tag parameter usage is described in SetRemoteTag (see page 656).

One should not change the Remote tag with this method once a dialog is established as with the SetRemoteTag (see page 656) method. Other parameters can be changed but be aware that changing such parameters could lead to interoperability problems with non-compliant peers.

This parameter list does not need to be configured by the application. It is automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

This parameter list can be obtained through GetRemoteDescriptorParameters (see page 647).

See Also

[SetRemoteTag](#) (see page 656), [GetRemoteDescriptorParameters](#) (see page 647)

3.7.1.42.1.32 - ISipUserAgentSvc::SetRemoteSequenceNumber Method

Sets the sequence number expected in the next request received.

C++

```
virtual void SetRemoteSequenceNumber(IN uint32_t uSequenceNumber) = 0;
```

Parameters

Parameters	Description
IN uint32_t uSequenceNumber	The expected remote sequence number.

Description

Sets the sequence number that is expected in the CSeq of the next request received on this dialog.

This parameter has meaning only when the state is eIN_DIALOG. When the state is eOUT_OF_DIALOG, it has absolutely no effect. When the state is eIN_DIALOG, the user agent service automatically rejects requests for which the CSeq is smaller than or equal to this parameter.

The application does not need to set this parameter. By default, it is initialized with the value in the CSeq header of the packet in parameter to Establish (see page 642), when this method is called with a received request or with the CSeq value of the first request received when eIN_DIALOG is true.

If one changes its value after it has been initialized, it could lead the application to either accept out of order requests from the remote peer or reject its valid requests as out of order requests.

This value can be obtained through GetRemoteSequenceNumber (see page 648).

See Also

[Establish](#) (see page 642), [GetRemoteSequenceNumber](#) (see page 648)

3.7.1.42.1.33 - ISipUserAgentSvc::SetRemoteSequenceNumber64bits Method

Sets the sequence number expected in the next request received.

C++

```
virtual void SetRemoteSequenceNumber64bits(IN uint64_t uSequenceNumber) = 0;
```

Parameters

Parameters	Description
IN uint64_t uSequenceNumber	The expected remote sequence number.

Description

Sets the sequence number that is expected in the CSeq of the next request received on this dialog.

This parameter has meaning only when the state is eIN_DIALOG. When the state is eOUT_OF_DIALOG it has absolutely no effect. When the state is eIN_DIALOG, the user agent service automatically rejects requests for which the CSeq is smaller than or equal to this parameter.

The application does not need to set this parameter. By default it is initialized with the value in the CSeq header of the packet in parameter to Establish (see page 642) when this method is called with a received request or with the CSeq value of the first request received when eIN_DIALOG is true.

If one changes its value after it has been initialized, it could lead the application to either accept out of order requests from the remote peer or reject its valid requests as out of order requests.

This value can be obtained through GetRemoteSequenceNumber (see page 648).

See Also

Establish (see page 642), GetRemoteSequenceNumber (see page 648)

3.7.1.42.1.34 - ISipUserAgentSvc::SetRemoteTag Method

Sets the tag parameter to put in the To header of requests sent.

C++

```
virtual void SetRemoteTag(IN const CString& rstrTag) = 0;
```

Parameters

Parameters	Description
IN const CString& rstrTag	The tag parameter to put in the To header of requests sent.

Description

Sets the tag parameter to put in the To header of requests created with CreateBasicRequest.

This parameter is also used to match the dialog for incoming packets when the user agent service is in the eIN_DIALOG state. When in such a state, its value must match the tag of the To header of incoming responses or the tag of the From header of incoming requests. If it does not match, the dialog does not match and the packet must be handled by another context.

The exception to this matching rule is when a forking request is sent and a forked packet is received. In this case, and when the dialog state is in eOUT_OF_DIALOG, the remote tag parameter is not used to match the dialog.

One should not change this parameter after a dialog has been established on this user agent service as it has the effect of breaking the dialog matching mechanism both at the local application and the remote peer.

This parameter does not need to be configured by the application. It is automatically updated with the information in the parameter to Establish (see page 642) when this method is called.

This parameter can be obtained through GetRemoteTag (see page 649). It is also contained in the parameters returned by GetRemoteDescriptorParameters (see page 647) unless it is an empty string.

See Also

GetRemoteTag (see page 649), GetRemoteDescriptorParameters (see page 647), CreateBasicRequest, Establish (see page 642).

3.7.1.42.1.35 - ISipUserAgentSvc::TerminateUsage Method

Terminates a usage of a dialog.

C++

```
virtual mxt_result TerminateUsage() = 0;
```

Returns

- resFE_INVALID_STATE: There is no dialog established in this context.

- **resSW_DIALOG_ALREADY_UNREGISTERED:** Processing has been done but the dialog was already unregistered from the dialog matcher list. This happens only when Clear was called on the Context.
- **resS_OK:** The dialog usage has been successfully terminated. Note that there could still be other usages, hence it does not mean that the dialog is terminated.

Description

This method must be called when a usage of a dialog is terminated. It could be when a final negative response for an INVITE is received after a provisional response that established a dialog or when a BYE request is received to name a few.

This method MUST be called exactly once for each call to Establish (see page 642). Calling it more often has an error returned. Calling it less often would mean that a dialog was established but never terminated. This is not an acceptable usage of dialogs.

See Also

Establish (see page 642)

3.7.1.42.1.36 - ISipUserAgentSvc::UpdateRoute Method

Updates the Route set of a dialog.

C++

```
virtual mxt_result UpdateRoute(IN const CSipPacket& rPacket) = 0;
```

Parameters

Parameters	Description
IN const CSipPacket& rPacket	The packet that refreshes the Route.

Returns

- **resFE_INVALID_STATE:** There is no dialog established in this context.
- **resFE_INVALID_ARGUMENT:** The packet in parameter is not a 2xx response to an INVITE.
- **resS_OK:** The Route has been properly updated.

Description

This method updates the Route set with the Record-Route headers found in the packet in parameter. This method only makes sense when receiving a 2xx to an INVITE.

In most cases, the Route cannot be changed once a dialog is established and, since Establish (see page 642) already takes care of updating the Route set, there is no need to call this method. The only exception is when an early dialog becomes confirmed as per RF C3261 terms. This object makes no distinction between early and confirmed dialogs. The dialog is established when it enters the early state and it is still established when it becomes confirmed. However, since on this occasion the Route set can be updated, a method to do so is needed. Hence, this method should only be called when a dialog transitions from early to confirmed.

It is invalid to call this method if the dialog state is eOUT_OF_DIALOG. In that case, only the Preloaded Route is used and the Route has no meaning.

The application does not need to bother with this as it is automatically handled by the ISipSessionSvc (see page 578).

See Also

Establish (see page 642), GetState (see page 649), ISipSessionSvc (see page 578)

3.7.1.42.2 - Enumerations

3.7.1.42.2.1 - ISipUserAgentSvc::EPacketDirection Enumeration

```
enum EPacketDirection {
    eRECEIVED,
    eSENT
};
```

Description

Indicate if it's a received or a sent packet.

Members

Members	Description
eRECEIVED	Packet was received.
eSENT	Packet was sent.

3.7.1.42.2.2 - ISipUserAgentSvc::EState Enumeration

```
enum EState {
    eIN_DIALOG,
    eWAITING_FOR_DIALOG_COMPLETION,
    eOUT_OF_DIALOG
};
```

Description

Indicate the state of the dialog.

Members

Members	Description
eIN_DIALOG	There is at least one active dialog usage on this context.
eWAITING_FOR_DIALOG_COMPLETION	Sent a request that permits the peer to create the dialog with a request.
eOUT_OF_DIALOG	There is no dialog usage on this context.

3.7.1.42.2.3 - ISipUserAgentSvc::ETargetHeaderOption Enumeration

This enum indicates if the target headers (of the actual current target that will be overwritten by the new current target) must be preserved and appended to the target headers of the new current target.

C++

```
enum ETargetHeaderOption {
    ePRESERVE_TARGET_HEADERS,
    eCLEAR_TARGET_HEADERS
};
```

Members

Members	Description
ePRESERVE_TARGET_HEADERS	The target headers of the actual current target are appended to the target headers of the new current target.
eCLEAR_TARGET_HEADERS	The target headers of the actual current target are cleared.

3.7.2 - Variables

This section documents the variables of the Sources/SipUserAgent folder.

3.7.2.1 - Sip Event Constants

This file contains the SIP Event names and their respective default expiration time in seconds. These are used in the subscriber and notifier services.

Location

SipUserAgent/SipEventConstants.h

3.7.2.1.1 - RFC 3842 Subscription Events

```
const char* const szSIP_EVENT_MESSAGE_SUMMARY = "message-summary";
const unsigned int uSIP_EVENT_MESSAGE_SUMMARY_DEFAULT_S = 3600;
```

Description

Subscription events defined in RFC 3842 and related constant.

3.7.2.1.2 - RFC 3856 Subscription Events

```
const char* const szSIP_EVENT_PRESENCE = "presence";
const unsigned int uSIP_EVENT_PRESENCE_DEFAULT_S = 3600;
```

Description

Subscription events defined in RFC 3856 and related constant.

3.7.2.1.3 - RFC 3515 Subscription Events

```
const char* const szSIP_EVENT_REFER = "refer";
const unsigned int uSIP_EVENT_REFER_DEFAULT_S = 60;
```

Description

Subscription events defined in RFC 3515 and related constant.

Note that this event is special since it is always the notifier that decides the subscription duration. 60 seconds should be long enough for the notifier to send its initial NOTIFY.

3.7.2.1.4 - RFC 3680 Subscription Events

```
const char* const szSIP_EVENT_REG = "reg";
const unsigned int uSIP_EVENT_REG_DEFAULT_S = 3761;
```

Description

Subscription events defined in RFC 3680 and related constant.

3.7.2.1.5 - RFC 3910 Subscription Events

```
const char* const szSIP_EVENT_SPIRITS_INDPS = "spirits-INDPs";
const unsigned int uSIP_EVENT_SPIRITS_INDPS_DEFAULT_S = 3600;
const char* const szSIP_EVENT_SPIRITS_USER_PROF = "spirits-user-prof";
const unsigned int uSIP_EVENT_SPIRITS_USER_PROF_DEFAULT_S = 3600;
```

Description

Subscription events defined in RFC 3910 and related constant.

Note that nothing is indicated as a default expiration time.

3.7.2.1.6 - RFC 3857 Subscription Events

```
const char* const szSIP_EVENT_WINFO = "winfo";
const unsigned int uSIP_EVENT_WINFO_DEFAULT_S = 3600;
```

Description

Subscription events defined in RFC 3857 and related constant.

3.8 - Startup

This section documents the Sources/Startup folder of the M5T SIP SAFE stack. It is divided in functional subsections:

- Classes (see page 660)

3.8.1 - Classes

This section documents the classes of the Sources/Startup folder.

Classes

Class	Description
CSipStackInitializer (see page 660)	

3.8.1.1 - CSipStackInitializer Class

Class Hierarchy

CSipStackInitializer

C++

```
class CSipStackInitializer;
```

Description

Initializes the SIP stack and its dependencies.

Location

Startup/CSipStackInitializer.h

Methods

Method	Description
Finalize (see page 660)	Finalizes the SIP stack and its dependencies.
Initialize (see page 660)	Initializes the SIP stack and its dependencies.
RegisterTracingNodes (see page 661)	Registers the module's tracing nodes.
UnregisterTracingNodes (see page 661)	Unregisters the module's tracing nodes.

Legend

Method

3.8.1.1.1 - Methods

3.8.1.1.1.1 - CSipStackInitializer::Finalize Method

Finalizes the SIP stack and its dependencies.

C++

```
static void Finalize();
```

Description

Finalizes the SIP stack and its dependencies. The application must call no method on any other classes or objects of the SIP stack after this method is called. This method must be called only from one thread and must not be called from a different thread than the Initialize (see page 660) method was called.

3.8.1.1.1.2 - CSipStackInitializer::Initialize Method

Initializes the SIP stack and its dependencies.

C++

```
static mxt_result Initialize();
```

Returns

- `resS_OK`: The SIP stack and its dependencies have been properly initialized. The stack needs to be Finalized when it is no longer needed.
- `resFE_FAIL`: One or more component failed to be initialized.

Description

Initializes the SIP stack and its dependencies. This method must be called before any method on any other classes or objects of the SIP stack. This method must be called only from one thread.

3.8.1.1.1.3 - CSipStackInitializer::RegisterTracingNodes Method

Registers the module's tracing nodes.

C++

```
static void RegisterTracingNodes();
```

Description

This method registers the module's tracing nodes.

3.8.1.1.1.4 - CSipStackInitializer::UnregisterTracingNodes Method

Unregisters the module's tracing nodes.

C++

```
static void UnregisterTracingNodes();
```

Description

This method unregisters the module's tracing nodes.

Index

C

CAbsoluteUri 128
 CAbsoluteUri class 128
 ~CAbsoluteUri 130
 = 133
 CAbsoluteUri 129, 130
 GenerateCopy 130
 GetBody 130
 GetScheme 131
 GetUriType 131
 IsEquivalent 131
 Parse 132
 Reset 132
 Serialize 132
 SetScheme 133
 CAbsoluteUri::~CAbsoluteUri 130
 CAbsoluteUri::= 133
 CAbsoluteUri::CAbsoluteUri 129, 130
 CAbsoluteUri::GenerateCopy 130
 CAbsoluteUri::GetBody 130
 CAbsoluteUri::GetScheme 131
 CAbsoluteUri::GetUriType 131
 CAbsoluteUri::IsEquivalent 131
 CAbsoluteUri::Parse 132
 CAbsoluteUri::Reset 132
 CAbsoluteUri::Serialize 132
 CAbsoluteUri::SetScheme 133
 CDate 133
 CDate class 133
 != 139
 ~CDate 136
 = 139
 == 140
 CDate 136
 GetDateTime 136
 GetDayOfMonth 136
 GetDayOfWeek 137
 GetHours 137
 GetMinutes 137
 GetMonth 137
 GetSeconds 138
 GetYear 138
 IsSet 138
 ms_szDATE_GMT 135
 ms_szDAY_FRIDAY 135
 ms_szDAY_MONDAY 135
 ms_szDAY_SATURDAY 135
 ms_szDAY_SUNDAY 135
 ms_szDAY_THURSDAY 135
 ms_szDAY_TUESDAY 135
 ms_szDAY_WEDNESDAY 135
 ms_szMONTH_APRIIL 135
 ms_szMONTH_AUGUST 135
 ms_szMONTH_DECEMBER 135
 ms_szMONTH_FEBRUARY 135
 ms_szMONTH_JANUARY 135
 ms_szMONTH_JULY 135
 ms_szMONTH_JUNE 135
 ms_szMONTH_MARCH 135
 ms_szMONTH_MAY 135
 ms_szMONTH_NOVEMBER 135
 ms_szMONTH_OCTOBER 135
 ms_szMONTH_SEPTEMBER 135
 Parse 138
 Reset 139
 Serialize 139
 CDate::!= 139
 CDate::~CDate 136
 CDate::= 139
 CDate::== 140
 CDate::CDate 136
 CDate::GetDateTime 136
 CDate::GetDayOfMonth 136
 CDate::GetDayOfWeek 137
 CDate::GetHours 137
 CDate::GetMinutes 137
 CDate::GetMonth 137

CDate::GetSeconds 138
CDate::GetYear 138
CDate::IsSet 138
CDate::ms_szDATE_GMT 135
CDate::ms_szDAY_FRIDAY 135
CDate::ms_szDAY_MONDAY 135
CDate::ms_szDAY_SATURDAY 135
CDate::ms_szDAY_SUNDAY 135
CDate::ms_szDAY_THURSDAY 135
CDate::ms_szDAY_TUESDAY 135
CDate::ms_szDAY_WEDNESDAY 135
CDate::ms_szMONTH_APRIl 135
CDate::ms_szMONTH_AUGUST 135
CDate::ms_szMONTH_DECEMBER 135
CDate::ms_szMONTH_FEBRUARY 135
CDate::ms_szMONTH_JANUARY 135
CDate::ms_szMONTH_JULY 135
CDate::ms_szMONTH_JUNE 135
CDate::ms_szMONTH_MARCH 135
CDate::ms_szMONTH_MAY 135
CDate::ms_szMONTH_NOVEMBER 135
CDate::ms_szMONTH_OCTOBER 135
CDate::ms_szMONTH_SEPTEMBER 135
CDate::Parse 138
CDate::Reset 139
CDate::SDateTime 135
CDate::SDateTime struct 135
CDate::Serialize 139
CGenericParam 140
CGenericParam class 140
!= 145
~CGenericParam 143
= 145
== 146
CGenericParam 142
GetName 143
GetValue 143
Parse 144
Reset 144
Serialize 144
ViaBranchStartsWithMagicCookie 145
CGenericParam::!= 145
CGenericParam::~CGenericParam 143
CGenericParam::= 145
CGenericParam::== 146
CGenericParam::CGenericParam 142
CGenericParam::ECharSet 146
CGenericParam::ECharSet enumeration 146
CGenericParam::GetName 143
CGenericParam::GetValue 143
CGenericParam::Parse 144
CGenericParam::Reset 144
CGenericParam::Serialize 144
CGenericParam::ViaBranchStartsWithMagicCookie 145
CGenParamList 146
CGenParamList class 146
!= 152
[] 152, 153
~CGenParamList 149
= 153
== 154
Append 149
CGenParamList 148
IsEmpty 149
Length 150
Parse 150
Remove 150, 151
Reset 151
Serialize 151
Set 152
CGenParamList::!= 152
CGenParamList::[] 152, 153
CGenParamList::~CGenParamList 149
CGenParamList::= 153
CGenParamList::== 154
CGenParamList::Append 149
CGenParamList::CGenParamList 148
CGenParamList::EStartWithSeparator 154
CGenParamList::EStartWithSeparator enumeration 154
CGenParamList::IsEmpty 149

CGenParamList::Length 150
CGenParamList::Parse 150
CGenParamList::Remove 150, 151
CGenParamList::Reset 151
CGenParamList::Serialize 151
CGenParamList::Set 152
CHeaderList 155
CHeaderList class 155
!= 166
~CHeaderList 156
= 166
== 166
Append 157
AppendRawData 157
CHeaderList 156
CommitRawDataList 158
Get 159, 160, 161
GetNbHeaderTypes 161
GetRawDataList 162
IsEmpty 162
Prepend 162
RemoveHeader 163
RemoveHeaderType 163, 164
ReplaceHeaders 164
ReplaceHeaderTypeWith 164
Reset 165
Serialize 165
Sort 165
CHeaderList::!= 166
CHeaderList::~CHeaderList 156
CHeaderList::= 166
CHeaderList::== 166
CHeaderList::Append 157
CHeaderList::AppendRawData 157
CHeaderList::CHeaderList 156
CHeaderList::CommitRawDataList 158
CHeaderList::Get 159, 160, 161
CHeaderList::GetNbHeaderTypes 161
CHeaderList::GetRawDataList 162
CHeaderList::IsEmpty 162
CHeaderList::Prepend 162
CHeaderList::RemoveHeader 163
CHeaderList::RemoveHeaderType 163, 164
CHeaderList::ReplaceHeaders 164
CHeaderList::ReplaceHeaderTypeWith 164
CHeaderList::Reset 165
CHeaderList::Serialize 165
CHeaderList::Sort 165
CHostPort 167
CHostPort class 167
!= 175
~CHostPort 169
= 176
== 176
CHostPort 168, 169
GetHost 169
GetPort 170
Ipv6AddressToIpv6Reference 170
Ipv6ReferenceToIpv6Address 170
IsHostName 171
IsIpv4Address 172
IsIpv6Reference 172, 173
ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT 168
Parse 174
Reset 175
Serialize 175
SetHost 175
CHostPort::!= 175
CHostPort::~CHostPort 169
CHostPort::= 176
CHostPort::== 176
CHostPort::CHostPort 168, 169
CHostPort::GetHost 169
CHostPort::GetPort 170
CHostPort::Ipv6AddressToIpv6Reference 170
CHostPort::Ipv6ReferenceToIpv6Address 170
CHostPort::IsHostName 171
CHostPort::IsIpv4Address 172
CHostPort::IsIpv6Reference 172, 173
CHostPort::ms_uUSE_SIP_DEFAULT_DONT_OUTPUT_PORT 168

168
CHostPort::Parse 174
CHostPort::Reset 175
CHostPort::Serialize 175
CHostPort::SetHost 175
CImUri 177
CImUri class 177
 ~CImUri 180
 CImUri 179
 GenerateCopy 180
 GetScheme 180
 GetUriType 180
 Parse 181
 Serialize 181
CImUri::~CImUri 180
CImUri::CImUri 179
CImUri::GenerateCopy 180
CImUri::GetScheme 180
CImUri::GetUriType 180
CImUri::Parse 181
CImUri::Serialize 181
CInstanceTracker 440
CInstanceTracker class 440
 DebugGetContextDump 443
 DebugGetContextTable 443
 DebugGetNumContextsLeft 443
 DebugGetNumPacketsLeft 444
 DebugGetNumRequestContextsLeft 444
 DebugGetNumTransactionsLeft 444
 DebugGetPacketDump 444
 DebugGetPacketTable 445
 DebugGetRequestContextDump 445
 DebugGetRequestContextTable 446
 DebugGetTransactionDump 446
 DebugGetTransactionTable 446
CInstanceTracker::DebugGetContextDump 443
CInstanceTracker::DebugGetContextTable 443
CInstanceTracker::DebugGetNumContextsLeft 443
CInstanceTracker::DebugGetNumPacketsLeft 444
CInstanceTracker::DebugGetNumRequestContextsLeft 444
CInstanceTracker::DebugGetNumTransactionsLeft 444
CInstanceTracker::DebugGetPacketDump 444
CInstanceTracker::DebugGetPacketTable 445
CInstanceTracker::DebugGetRequestContextDump 445
CInstanceTracker::DebugGetRequestContextTable 446
CInstanceTracker::DebugGetTransactionDump 446
CInstanceTracker::DebugGetTransactionTable 446
CInstanceTracker::SMemoryRecords 441
CInstanceTracker::SMemoryRecords struct 441
 ~SMemoryRecords 442
 m_astMemoryBlocks 442
 m_szTypeName 442
 m_uNumberOfAllocatedMemoryBlocks 442
 SMemoryRecords 442
CInstanceTracker::SMemoryRecords::~SMemoryRecords 442
CInstanceTracker::SMemoryRecords::m_astMemoryBlocks 442
CInstanceTracker::SMemoryRecords::m_szTypeName 442
CInstanceTracker::SMemoryRecords::m_uNumberOfAllocatedMemoryBlocks 442
CInstanceTracker::SMemoryRecords::SMemoryRecords 442
Classes 18, 78, 127, 402, 440, 471, 660
CMailboxUri 182
CMailboxUri class 182
 != 190
 ~CMailboxUri 185
 = 191
 == 191
 CMailboxUri 184
 GenerateCopy 185
 GetDisplayName 185
 GetHeaderList 186
 GetHostPort 186
 GetScheme 186
 GetUriType 187
 GetUser 187, 332
 IsEquivalent 187
 Parse 188
 Reset 189
 Serialize 189

Set 189
 SetDisplayName 190
 SetHeaderList 190
 CMailboxUri::!= 190
 CMailboxUri::~CMailboxUri 185
 CMailboxUri::= 191
 CMailboxUri::== 191
 CMailboxUri::CMailboxUri 184
 CMailboxUri::GenerateCopy 185
 CMailboxUri::GetDisplayName 185
 CMailboxUri::GetHeaderList 186
 CMailboxUri::GetHostPort 186
 CMailboxUri::GetScheme 186
 CMailboxUri::GetUriType 187
 CMailboxUri:: GetUser 187, 332
 CMailboxUri::IsEquivalent 187
 CMailboxUri::Parse 188
 CMailboxUri::Reset 189
 CMailboxUri::Serialize 189
 CMailboxUri::Set 189
 CMailboxUri::SetDisplayName 190
 CMailboxUri::SetHeaderList 190
 CMessageSummary 192
 CMessageSummary class 192
 ~CMessageSummary 197
 = 201
 CMessageSummary 197
 GetHeaderList 198
 GetMsgAccount 198
 GetMsgStatus 198
 GetMsgSummaryLines 199
 ms_szMWI_MESSAGE_ACCOUNT 194
 ms_szMWI_MESSAGES_WAITING 194
 ms_szMWI_MSG_CONTEXT_CLASS_FAX 194
 ms_szMWI_MSG_CONTEXT_CLASS_MULTIMEDIA 194
 ms_szMWI_MSG_CONTEXT_CLASS_NONE 194
 ms_szMWI_MSG_CONTEXT_CLASS_PAGER 194
 ms_szMWI_MSG_CONTEXT_CLASS_TEXT 195
 ms_szMWI_MSG_CONTEXT_CLASS_VOICE 195
 ms_szMWI_NO 195
 ms_szMWI_YES 195
 ms_uMWI_MESSAGE_ACCOUNT_LEN_IN_BYTES 195
 ms_uMWI_MESSAGES_WAITING_LEN_IN_BYTES 195
 Parse 199
 Reset 200
 Serialize 200
 SetMsgAccount 200
 SetMsgStatus 200
 CMessageSummary::~CMessageSummary 197
 CMessageSummary::= 201
 CMessageSummary::CMessageSummary 197
 CMessageSummary::GetHeaderList 198
 CMessageSummary::GetMsgAccount 198
 CMessageSummary::GetMsgStatus 198
 CMessageSummary::GetMsgSummaryLines 199
 CMessageSummary::ms_szMWI_MESSAGE_ACCOUNT 194
 CMessageSummary::ms_szMWI_MESSAGES_WAITING 194
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_FAX 194
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_MULTI
 MEDIA 194
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_NONE 194
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_PAGE
 R 194
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_TEXT 195
 CMessageSummary::ms_szMWI_MSG_CONTEXT_CLASS_VOICE 195
 CMessageSummary::ms_szMWI_NO 195
 CMessageSummary::ms_szMWI_YES 195
 CMessageSummary::ms_uMWI_MESSAGE_ACCOUNT_LEN_IN
 BYTES 195
 CMessageSummary::ms_uMWI_MESSAGES_WAITING_LEN_IN
 BYTES 195
 CMessageSummary::Parse 199
 CMessageSummary::Reset 200
 CMessageSummary::Serialize 200
 CMessageSummary::SetMsgAccount 200

CMessageSummary::SetMsgStatus 200
 CMessageSummary::SSummaryLine 195
 CMessageSummary::SSummaryLine struct 195
 m_strMessageContextClass 196
 m_uNewMsgs 196
 m_uNewUrgentMsgs 196
 m_uOldMsgs 196
 m_uOldUrgentMsgs 197
 SSummaryLine 197
 CMessageSummary::SSummaryLine::m_strMessageContextClass 196
 CMessageSummary::SSummaryLine::m_uNewMsgs 196
 CMessageSummary::SSummaryLine::m_uNewUrgentMsgs 196
 CMessageSummary::SSummaryLine::m_uOldMsgs 196
 CMessageSummary::SSummaryLine::m_uOldUrgentMsgs 197
 CMessageSummary::SSummaryLine::SSummaryLine 197
 CNameAddr 201
 CNameAddr class 201
 != 208
 ~CNameAddr 203
 = 209
 == 209, 210
 CNameAddr 203
 GetDisplayName 204
 GetMailboxUri 204
 GetSipUri 204, 205
 GetUri 205
 Parse 205
 Reset 206
 Serialize 206
 SetMailboxUri 207
 SetSipUri 207
 SetUri 207
 CNameAddr::!= 208
 CNameAddr::~CNameAddr 203
 CNameAddr::= 209
 CNameAddr::== 209, 210
 CNameAddr::CNameAddr 203
 CNameAddr::EAngleBracket 210
 CNameAddr::EAngleBracket enumeration 210
 CNameAddr::GetDisplayName 204
 CNameAddr::GetMailboxUri 204
 CNameAddr::GetSipUri 204, 205
 CNameAddr::GetUri 205
 CNameAddr::Parse 205
 CNameAddr::Reset 206
 CNameAddr::Serialize 206
 CNameAddr::SetMailboxUri 207
 CNameAddr::SetSipUri 207
 CNameAddr::SetUri 207
 Config 5
 Configuring the SIP Stack with "PreSipStackCfg.h" 5
 CPresUri 211
 CPresUri class 211
 ~CPresUri 213
 CPresUri 213
 GenerateCopy 213
 GetScheme 214
 GetUriType 214
 Parse 214
 Serialize 214
 CPresUri::~CPresUri 213
 CPresUri::CPresUri 213
 CPresUri::GenerateCopy 213
 CPresUri::GetScheme 214
 CPresUri::GetUriType 214
 CPresUri::Parse 214
 CPresUri::Serialize 214
 CQuotedString 215
 CQuotedString class 215
 ~CQuotedString 216
 = 218
 CQuotedString 216
 GetString 217
 Parse 217
 Reset 217
 Serialize 217
 CQuotedString::~CQuotedString 216
 CQuotedString::= 218
 CQuotedString::CQuotedString 216

CQuotedString::GetString 217
CQuotedString::Parse 217
CQuotedString::Reset 217
CQuotedString::Serialize 217
CRawHeader 218
CRawHeader class 218
 ~CRawHeader 220
 = 222
 AppendRawData 220
 CRawHeader 219
 GetBody 221
 GetName 221
 IsComplete 221
 Reset 222
CRawHeader::~CRawHeader 220
CRawHeader::= 222
CRawHeader::AppendRawData 220
CRawHeader::CRawHeader 219
CRawHeader::GetBody 221
CRawHeader::GetName 221
CRawHeader::IsComplete 221
CRawHeader::Reset 222
CRawHeader::Reset 222
CReginfo 222
CReginfo class 222
 ~CReginfo 229
 AppendRegistration 230
 CReginfo 229
 GetRegistration 230
 GetRegistrationCount 230
 GetState 231
 GetVersion 231
 Parse 231, 232
 Reset 232
 Serialize 232
 SetState 233
 SetVersion 233
CReginfo::~CReginfo 229
CReginfo::AppendRegistration 230
CReginfo::CReginfo 229
CReginfo::GetRegistration 230
CReginfo::GetRegistrationCount 230
CReginfo::GetState 231
CReginfo::GetVersion 231
CReginfo::Parse 231, 232
CReginfo::Reset 232
CReginfo::SContact 223
CReginfo::SContact struct 223
 ~SContact 227
 IsValid 227
 m_nameAddr 224
 m_pPubGruu 224
 m_pTempGruu 224
 m_strCallId 225
 m_strEvent 225
 m_strId 225
 m_strInstancId 225
 m_strQ 225
 m_strState 226
 m_uCSeq 226
 m_uDurationRegisteredS 226
 m_uExpiresS 226
 m_unknownParams 226
 m_uRetryAfterS 227
 SContact 227
CReginfo::SContact::~SContact 227
CReginfo::SContact::IsValid 227
CReginfo::SContact::m_nameAddr 224
CReginfo::SContact::m_pPubGruu 224
CReginfo::SContact::m_pTempGruu 224
CReginfo::SContact::m_strCallId 225
CReginfo::SContact::m_strEvent 225
CReginfo::SContact::m_strId 225
CReginfo::SContact::m_strInstancId 225
CReginfo::SContact::m_strQ 225
CReginfo::SContact::m_strState 226
CReginfo::SContact::m_uCSeq 226
CReginfo::SContact::m_uDurationRegisteredS 226
CReginfo::SContact::m_uExpiresS 226
CReginfo::SContact::m_unknownParams 226
CReginfo::SContact::m_uRetryAfterS 227

CReginfo::SContact::SContact 227
CReginfo::Serialize 232
CReginfo::SetState 233
CReginfo::SetVersion 233
CReginfo::SRegistration 227
CReginfo::SRegistration struct 227
 ~SRegistration 229
 IsValid 229
 m_pAor 228
 m_strId 228
 m_strState 228
 m_vecpstContacts 228
 SRegistration 229
CReginfo::SRegistration::~SRegistration 229
CReginfo::SRegistration::IsValid 229
CReginfo::SRegistration::m_pAor 228
CReginfo::SRegistration::m_strId 228
CReginfo::SRegistration::m_strState 228
CReginfo::SRegistration::m_vecpstContacts 228
CReginfo::SRegistration::SRegistration 229
CRequestLine 234
CRequestLine class 234
 ~CRequestLine 235
 = 238
 CRequestLine 235
 GetMethod 235
 GetMethodToken 236
 GetSipUri 236
 GetUri 236
 Parse 237
 Reset 237
 Serialize 237
 SetRequestUri 238
 SetSipUri 238
CRequestLine::~CRequestLine 235
CRequestLine::= 238
CRequestLine::CRequestLine 235
CRequestLine::GetMethod 235
CRequestLine::GetMethodToken 236
CRequestLine::GetSipUri 236
CRequestLine::GetUri 236
CRequestLine::Parse 237
CRequestLine::Reset 237
CRequestLine::Serialize 237
CRequestLine::SetRequestUri 238
CRequestLine::SetSipUri 238
CSipHeader 239
CSipHeader class 239
 != 296
 ~CSipHeader 245
 = 296
 == 296
 AppendNextHeader 246
 CopySingleHeader 246
 CSipHeader 245
 GetAcceptContact 246
 GetAcceptEncoding 247
 GetAcceptLanguage 247
 GetAcceptMSubType 248
 GetAcceptMType 248
 GetAcceptResourcePriority 248
 GetAlertInfo 249
 GetAllow 249
 GetAllowEvents 250
 GetAuthenticationInfo 250
 GetAuthorizationScheme 250
 GetCallId 251
 GetCallInfo 251
 GetContact 252
 GetContentDescription 252
 GetContentDisposition 252
 GetContentEncoding 253
 GetContentId 253
 GetContentLanguage 254
 GetContentLength 254
 GetContentTransferEncoding 254
 GetContentTypeMSubType 255
 GetContentTypeMType 255
 GetCSeqMethod 256
 GetCSeqNumber 256

GetDate 256
GetDiversion 257
GetErrorInfo 256, 257
GetEvent 257
GetExpires 258
GetExtensionHeaderName 258
GetExtensionHeaderValue 259
GetFlowTimer 258, 259
GetFrom 259
GetHeaderName 260
GetHeaderType 260
GetHistoryInfo 260
GetIdentity 261
GetIdentityInfo 261
GetInReplyTo 261
GetJoin 262
GetLongHeaderName 262
GetMaxForwards 263
GetMimeType 263
GetMinExpires 263
GetMinSe 264
GetNbNextHeaders 264
GetNbParsedHeaders 264
GetNextHeader 265
GetOrganization 265
GetPAccessNetworkInfo 266
GetParam 266
GetParamList 267
GetPAssertedIdentity 268
GetPAssociatedUri 268
GetPath 268
GetPCalledPartyId 269
GetPChargingFunctionAddresses 269
GetPChargingVector 270
GetPMediaAuthorization 270
GetPPREFERREDIdentity 270
GetPriority 271
GetPrivacy 271
GetProxyAuthenticateScheme 272
GetProxyAuthorizationScheme 272
GetProxyRequire 273
GetPVisitedNetworkId 273
GetRackCSeq 273
GetRackMethod 274
GetRackResponseNum 274
GetRawHeader 274, 275
GetReason 275
GetRecordRoute 275
GetReferredBy 276
GetReferTo 276
GetRejectContact 277
GetReplaces 277
GetReplyTo 278
GetRequestDisposition 278
GetRequire 278
GetResourcePriority 279
GetRetryAfter 279
GetRetryAfterComment 280
GetRoute 280
GetRSeq 280
GetServer 281
GetServiceRoute 281
GetSessionExpires 281
GetShortHeaderName 282
GetSipETag 282
GetSipIfMatch 283
GetSubject 283
GetSubscriptionState 283
GetSupported 284
GetTargetDialog 284
GetTimestamp 285
GetTo 285
GetUnsupported 285
GetUserAgent 286
GetViaProtocolName 286
GetViaProtocolVersion 287
GetViaSentBy 287
GetViaTransport 287
GetWarning 288
GetWwwAuthenticateScheme 288

InsertNextHeader 288	CSipHeader::GetAuthenticationInfo 250
IsContactWildcard 289	CSipHeader::GetAuthorizationScheme 250
IsEmptyHeader 289	CSipHeader::GetCallId 251
IsParsedDataAvailable 290	CSipHeader::GetCallInfo 251
IsSingleHdrEquivalent 290	CSipHeader::GetContact 252
Parse 290	CSipHeader::GetContentDescription 252
RemoveNextHeader 291	CSipHeader::GetContentDisposition 252
ReplaceNextHeader 292	CSipHeader::GetContentEncoding 253
Reset 292	CSipHeader::GetContentId 253
Serialize 292	CSipHeader::GetContentLanguage 254
SetContactWildcard 292	CSipHeader::GetContentLength 254
SetNameForm 293	CSipHeader::GetContentTransferEncoding 254
SetParam 293, 294	CSipHeader::GetContentTypeMSubType 255
SetParamList 294	CSipHeader::GetContentTypeMType 255
SetRawHeader 294	CSipHeader::GetCSeqMethod 256
UnlinkNextHeader 295	CSipHeader::GetCSeqNumber 256
UnlinkTopHeader 295	CSipHeader::GetDate 256
CSipHeader Constants 401	CSipHeader::GetDiversion 257
CSipHeader::!= 296	CSipHeader::GetErrorInfo 256, 257
CSipHeader::~CSipHeader 245	CSipHeader::GetEvent 257
CSipHeader::= 296	CSipHeader::GetExpires 258
CSipHeader::== 296	CSipHeader::GetExtensionHeaderName 258
CSipHeader::AppendNextHeader 246	CSipHeader::GetExtensionHeaderValue 259
CSipHeader::CopySingleHeader 246	CSipHeader::GetFlowTimer 258, 259
CSipHeader::CSipHeader 245	CSipHeader::GetFrom 259
CSipHeader::EHeaderNameForm 297	CSipHeader::GetHeaderName 260
CSipHeader::EHeaderNameForm enumeration 297	CSipHeader::GetHeaderType 260
CSipHeader::EParameterCreationBehavior 297	CSipHeader::GetHistoryInfo 260
CSipHeader::EParameterCreationBehavior enumeration 297	CSipHeader::GetIdentity 261
CSipHeader::EResetLevel 297	CSipHeader::GetIdentityInfo 261
CSipHeader::EResetLevel enumeration 297	CSipHeader::GetInReplyTo 261
CSipHeader::GetAcceptContact 246	CSipHeader::GetJoin 262
CSipHeader::GetAcceptEncoding 247	CSipHeader::GetLongHeaderName 262
CSipHeader::GetAcceptLanguage 247	CSipHeader::GetMaxForwards 263
CSipHeader::GetAcceptMSubType 248	CSipHeader::GetMimeVersion 263
CSipHeader::GetAcceptMType 248	CSipHeader::GetMinExpires 263
CSipHeader::GetAcceptResourcePriority 248	CSipHeader::GetMinSe 264
CSipHeader::GetAlertInfo 249	CSipHeader::GetNbNextHeaders 264
CSipHeader::GetAllow 249	CSipHeader::GetNbParsedHeaders 264
CSipHeader::GetAllowEvents 250	CSipHeader::GetNextHeader 265

CSipHeader::GetOrganization 265
CSipHeader::GetPAccessNetworkInfo 266
CSipHeader::GetParam 266
CSipHeader::GetParamList 267
CSipHeader::GetPAssertedIdentity 268
CSipHeader::GetPAssociatedUri 268
CSipHeader::GetPath 268
CSipHeader::GetPCalledPartyId 269
CSipHeader::GetPChargingFunctionAddresses 269
CSipHeader::GetPChargingVector 270
CSipHeader::GetPMediaAuthorization 270
CSipHeader::GetPPreferredIdentity 270
CSipHeader::GetPriority 271
CSipHeader::GetPrivacy 271
CSipHeader::GetProxyAuthenticateScheme 272
CSipHeader::GetProxyAuthorizationScheme 272
CSipHeader::GetProxyRequire 273
CSipHeader::GetPVisitedNetworkId 273
CSipHeader::GetRackCSeq 273
CSipHeader::GetRackMethod 274
CSipHeader::GetRackResponseNum 274
CSipHeader::GetRawHeader 274, 275
CSipHeader::GetReason 275
CSipHeader::GetRecordRoute 275
CSipHeader::GetReferredBy 276
CSipHeader::GetReferTo 276
CSipHeader::GetRejectContact 277
CSipHeader::GetReplaces 277
CSipHeader::GetReplyTo 278
CSipHeader::GetRequestDisposition 278
CSipHeader::GetRequire 278
CSipHeader::GetResourcePriority 279
CSipHeader::GetRetryAfter 279
CSipHeader::GetRetryAfterComment 280
CSipHeader::GetRoute 280
CSipHeader::GetRSeq 280
CSipHeader::GetServer 281
CSipHeader::GetServiceRoute 281
CSipHeader::GetSessionExpires 281
CSipHeader::GetShortHeaderName 282
CSipHeader::GetSipETag 282
CSipHeader::GetSipIfMatch 283
CSipHeader::GetSubject 283
CSipHeader::GetSubscriptionState 283
CSipHeader::GetSupported 284
CSipHeader::GetTargetDialog 284
CSipHeader::GetTimestamp 285
CSipHeader::GetTo 285
CSipHeader::GetUnsupported 285
CSipHeader::GetUserAgent 286
CSipHeader::GetViaProtocolName 286
CSipHeader::GetViaProtocolVersion 287
CSipHeader::GetViaSentBy 287
CSipHeader::GetViaTransport 287
CSipHeader::GetWarning 288
CSipHeader::GetWwwAuthenticateScheme 288
CSipHeader::InsertNextHeader 288
CSipHeader::IsContactWildcard 289
CSipHeader::IsEmptyHeader 289
CSipHeader::IsParsedDataAvailable 290
CSipHeader::IsSingleHdrEquivalent 290
CSipHeader::Parse 290
CSipHeader::RemoveNextHeader 291
CSipHeader::ReplaceNextHeader 292
CSipHeader::Reset 292
CSipHeader::Serialize 292
CSipHeader::SetContactWildcard 292
CSipHeader::SetNameForm 293
CSipHeader::SetParam 293, 294
CSipHeader::SetParamList 294
CSipHeader::SetRawHeader 294
CSipHeader::UnlinkNextHeader 295
CSipHeader::UnlinkTopHeader 295
CSipMessageBody 298
CSipMessageBody class 298
 ~CSipMessageBody 306
 = 311
 AddBody 306, 307
 class CSipPacketParser 311
 CSipMessageBody 306

- GetBlobBody 307, 308
 GetExternalMimeHeaders 308
 GetMimeHeaders 309
 GetNumberOfBodies 309
 GetSipMessageBody 309
 ms_pszBOUNDARY_OUTER_LEVEL 304
 ms_pszCONTENT_TYPE_APPLICATION_MEDIA_TYPE 304
 ms_pszCONTENT_TYPE_DIGEST_MEDIA_SUBTYPE 304
 ms_pszCONTENT_TYPE_MESSAGE_MEDIA_TYPE 304
 ms_pszCONTENT_TYPE_MIXED_MEDIA_SUBTYPE 304
 ms_pszCONTENT_TYPE_MULTIPART_MEDIA_TYPE 304
 ms_pszCONTENT_TYPE_PKCS7_SIGNATURE_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_PLAIN_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_RFC822_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_SDP_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_SIGNED_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_SIP_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_SIPFRAG_MEDIA_SUBTYPE 305
 ms_pszCONTENT_TYPE_TEXT_MEDIA_TYPE 306
 Reset 310
 Serialize 310
 SetExternalMimeHeaders 310
 CSipMessageBody::~CSipMessageBody 306
 CSipMessageBody::= 311
 CSipMessageBody::AddBody 306, 307
 CSipMessageBody::CSipMessageBody 306
 CSipMessageBody::GetBlobBody 307, 308
 CSipMessageBody::GetExternalMimeHeaders 308
 CSipMessageBody::GetMimeHeaders 309
 CSipMessageBody::GetNumberOfBodies 309
 CSipMessageBody::GetSipMessageBody 309
 CSipMessageBody::ms_pszBOUNDARY_OUTER_LEVEL 304
 CSipMessageBody::ms_pszCONTENT_TYPE_APPLICATION_MEDIA_TYPE 304
 CSipMessageBody::ms_pszCONTENT_TYPE_DIGEST_MEDIA_SUBTYPE 304
 CSipMessageBody::ms_pszCONTENT_TYPE_MESSAGE_MEDIA_TYPE 304
 CSipMessageBody::ms_pszCONTENT_TYPE_MIXED_MEDIA_SUBTYPE 304
 CSipMessageBody::ms_pszCONTENT_TYPE_MULTIPART_MEDIA_TYPE 304
 CSipMessageBody::ms_pszCONTENT_TYPE_PKCS7_SIGNATURE_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_PLAIN_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_RFC822_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_SDP_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_SIGNED_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_SIP_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_SIPFRAG_MEDIA_SUBTYPE 305
 CSipMessageBody::ms_pszCONTENT_TYPE_TEXT_MEDIA_TYPE 306
 CSipMessageBody::Reset 310
 CSipMessageBody::Serialize 310
 CSipMessageBody::SetExternalMimeHeaders 310
 CSipPacket 447
 CSipPacket class 447
 = 457
 AddRef 451
 AddServerToResponses 451
 AddUserAgentToRequests 452
 CSipPacket 449, 450
 GetLocalAddr 452
 GetMaxSize 452
 GetNextHopUri 452
 GetNextHopUriHostname 453
 GetPeerAddr 453
 GetTransport 453
 IsNewConnectionAllowed 453

IsReceivedOnAuthenticatedConnection 454	CSipPacketParser 312
IsStatHandledForReception 454	CSipPacketParser class 312
Release 454	~CSipPacketParser 314
SetAllowNewConnection 454	= 321
SetEntityId 455	AppendRawData 314
SetLocalAddr 455	CommitRawDataList 315
SetMaxForwards 455	CreateSipMessageBody 315
SetMaxSize 456	CSipPacketParser 313, 314
SetNextHopUri 456	GetHeaderList 316
SetPeerAddr 456	GetPayload 316
SetReceivedOnAuthenticatedConnection 457	GetRawDataList 316
SetStatHandledForReception 457	GetRequestLine 317
SetTransport 457	GetSipMessageBody 317, 318
CSipPacket::= 457	GetStatusLine 318
CSipPacket::AddRef 451	IsLocallyGenerated 318
CSipPacket::AddServerToResponses 451	IsRequest 318
CSipPacket::AddUserAgentToRequests 452	IsResponse 319
CSipPacket::CSipPacket 449, 450	Reset 319
CSipPacket::ERecordRouteProcessing 458	Serialize 319
CSipPacket::ERecordRouteProcessing enumeration 458	SetLocallyGenerated 319
CSipPacket::GetLocalAddr 452	SetPayload 320
CSipPacket::GetMaxSize 452	SetRequestLine 320
CSipPacket::GetNextHopUri 452	SetSipMessageBody 320
CSipPacket::GetNextHopUriHostname 453	SetStatusLine 320
CSipPacket::GetPeerAddr 453	CSipPacketParser::~CSipPacketParser 314
CSipPacket::GetTransport 453	CSipPacketParser::= 321
CSipPacket::IsNewConnectionAllowed 453	CSipPacketParser::AppendRawData 314
CSipPacket::IsReceivedOnAuthenticatedConnection 454	CSipPacketParser::CommitRawDataList 315
CSipPacket::IsStatHandledForReception 454	CSipPacketParser::CreateSipMessageBody 315
CSipPacket::Release 454	CSipPacketParser::CSipPacketParser 313, 314
CSipPacket::SetAllowNewConnection 454	CSipPacketParser::GetHeaderList 316
CSipPacket::SetEntityId 455	CSipPacketParser::GetPayload 316
CSipPacket::SetLocalAddr 455	CSipPacketParser::GetRawDataList 316
CSipPacket::SetMaxForwards 455	CSipPacketParser::GetRequestLine 317
CSipPacket::SetMaxSize 456	CSipPacketParser::GetSipMessageBody 317, 318
CSipPacket::SetNextHopUri 456	CSipPacketParser::GetStatusLine 318
CSipPacket::SetPeerAddr 456	CSipPacketParser::IsLocallyGenerated 318
CSipPacket::SetReceivedOnAuthenticatedConnection 457	CSipPacketParser::IsRequest 318
CSipPacket::SetStatHandledForReception 457	CSipPacketParser::IsResponse 319
CSipPacket::SetTransport 457	CSipPacketParser::Reset 319

CSipPacketParser::Serialize 319	GetNumFinalResponseRx 81
CSipPacketParser::SetLocallyGenerated 319	GetNumFinalResponseTx 81
CSipPacketParser::SetPayload 320	GetNumNonFinalResponseRetransmissionsTx 81
CSipPacketParser::SetRequestLine 320	GetNumNonFinalResponseRx 81
CSipPacketParser::SetSipMessageBody 320	GetNumNonFinalResponseTx 81
CSipPacketParser::SetStatusLine 320	GetNumRequestRetransmissionsRx 81
CSipPacketParser::SRawData 313	GetNumRequestRetransmissionsTx 81
CSipPacketParser::SRawData struct 313	GetNumRequestsRx 82
CSipProxyConfig 402	GetNumRequestsTx 82
CSipProxyConfig class 402	GetTotalNumRequestsRx 82
AddRef 403	GetTotalNumRequestsTx 82
CSipProxyConfig 402	GetTotalNumResponsesRx 82
GetAddresses 403	GetTotalNumResponsesTx 82
GetAuthenticationRealm 404	GetTotalNumTransactions 82
GetProxyId 404	NotifyDnsQueryResult 82
Release 405	NotifyReceivedPacket 83
CSipProxyConfig::AddRef 403	NotifySentPacket 83
CSipProxyConfig::CSipProxyConfig 402	NotifyTransactionEnd 83
CSipProxyConfig::GetAddresses 403	NotifyTransactionStart 83
CSipProxyConfig::GetAuthenticationRealm 404	Reset 83
CSipProxyConfig::GetProxyId 404	CSipStatisticsContainer::~CSipStatisticsContainer 80
CSipProxyConfig::Release 405	CSipStatisticsContainer::GetNumActiveTransactions 80
CSipStackInitializer 660	CSipStatisticsContainer::GetNumDnsQueriesFailed 80
CSipStackInitializer class 660	CSipStatisticsContainer::GetNumDnsQueriesSucceeded 80
Finalize 660	CSipStatisticsContainer::GetNumFinalResponseRetransmissionsRx
Initialize 660	81
RegisterTracingNodes 661	CSipStatisticsContainer::GetNumFinalResponseRetransmissionsTx
UnregisterTracingNodes 661	81
CSipStackInitializer::Finalize 660	CSipStatisticsContainer::GetNumFinalResponseRx 81
CSipStackInitializer::Initialize 660	CSipStatisticsContainer::GetNumFinalResponseTx 81
CSipStackInitializer::RegisterTracingNodes 661	CSipStatisticsContainer::GetNumNonFinalResponseRetransmissionsTx
CSipStackInitializer::UnregisterTracingNodes 661	81
CSipStatisticsContainer 78	CSipStatisticsContainer::GetNumNonFinalResponseRx 81
CSipStatisticsContainer class 78	CSipStatisticsContainer::GetNumNonFinalResponseTx 81
~CSipStatisticsContainer 80	CSipStatisticsContainer::GetNumRequestRetransmissionsRx 81
GetNumActiveTransactions 80	CSipStatisticsContainer::GetNumRequestRetransmissionsTx 81
GetNumDnsQueriesFailed 80	CSipStatisticsContainer::GetNumRequestsRx 82
GetNumDnsQueriesSucceeded 80	CSipStatisticsContainer::GetNumRequestsTx 82
GetNumFinalResponseRetransmissionsRx 81	CSipStatisticsContainer::GetTotalNumRequestsRx 82
GetNumFinalResponseRetransmissionsTx 81	CSipStatisticsContainer::GetTotalNumRequestsTx 82

CSipStatisticsContainer::GetTotalNumResponsesRx	82	GetHostPort	329
CSipStatisticsContainer::GetTotalNumResponsesTx	82	GetMissingPortBehavior	329
CSipStatisticsContainer::GetTotalNumTransactions	82	GetParam	330
CSipStatisticsContainer::NotifyDnsQueryResult	82	GetParamList	331
CSipStatisticsContainer::NotifyReceivedPacket	83	GetParamTransport	331
CSipStatisticsContainer::NotifySentPacket	83	GetPassword	331
CSipStatisticsContainer::NotifyTransactionEnd	83	GetScheme	331
CSipStatisticsContainer::NotifyTransactionStart	83	GetUriType	332
CSipStatisticsContainer::Reset	83	GetUser	332
CSipStatusLine	321	IsEquivalent	332
CSipStatusLine class	321	IsSecured	333
~CSipStatusLine	323	Parse	334
=	325	Reset	335
CSipStatusLine	322	Serialize	335
GetClass	323	Set	335
GetCode	323	SetHeaderList	335
GetPhrase	323	SetMissingPortBehavior	336
Parse	323	SetParam	336, 337
Reset	324	SetParamList	337
Serialize	324	SetPassword	337
Set	325	SetSecured	338
CSipStatusLine::~CSipStatusLine	323	CSipUri::!=	338
CSipStatusLine::=	325	CSipUri::~CSipUri	328
CSipStatusLine::CSipStatusLine	322	CSipUri::=	339
CSipStatusLine::GetClass	323	CSipUri::==	340
CSipStatusLine::GetCode	323	CSipUri::CSipUri	328
CSipStatusLine::GetPhrase	323	CSipUri::EParameterCreationBehavior	341
CSipStatusLine::Parse	323	CSipUri::EParameterCreationBehavior	enumeration 341
CSipStatusLine::Reset	324	CSipUri::ESecurityFlag	341
CSipStatusLine::Serialize	324	CSipUri::ESecurityFlag	enumeration 341
CSipStatusLine::Set	325	CSipUri::GenerateCopy	329
CSipUri	325	CSipUri::GetHeaderList	329
CSipUri class	325	CSipUri::GetHostPort	329
!=	338	CSipUri::GetMissingPortBehavior	329
~CSipUri	328	CSipUri::GetParam	330
=	339	CSipUri::GetParamList	331
==	340	CSipUri::GetParamTransport	331
CSipUri	328	CSipUri::GetPassword	331
GenerateCopy	329	CSipUri::GetScheme	331
GetHeaderList	329	CSipUri::GetUriType	332

CSipUri:: GetUser 332
CSipUri:: IsEquivalent 332
CSipUri:: IsSecured 333
CSipUri:: Parse 334
CSipUri:: Reset 335
CSipUri:: Serialize 335
CSipUri:: Set 335
CSipUri:: SetHeaderList 335
CSipUri:: SetMissingPortBehavior 336
CSipUri:: SetParam 336, 337
CSipUri:: SetParamList 337
CSipUri:: SetPassword 337
CSipUri:: SetSecured 338
CStringHelper 341
CStringHelper class 341
 AdaptForDisplay 343
 ConvertFromHexAscii 343
 ConvertToHexAscii 344
 EscapeChar 344
 GetEscaped 345
 IsAlpha 345
 IsAlphaNum 345
 IsDigit 346
 IsFloat 346
 IsHexadecimal 346
 IsLineTerminator 347
 IsLWS 347
 IsNumeric 348
 IsStringQdTextOrQuotedPair 348
 ms_cCR 342
 ms_cLF 342
 ms_cNUL 342
 ms_cSP 343
 ms_cTAB 343
 ms_pszCRLF 343
 ms_pszDASHBOUNDARY 343
 ms_pszDOUBLECRLF 343
 QuotedStringToString 348
 RemoveVisualSeparators 349
 SkipLWS 349
 SkipToData 349
 SkipWSP 350
 StringToQuotedString 350
 CStringHelper:: AdaptForDisplay 343
 CStringHelper:: ConvertFromHexAscii 343
 CStringHelper:: ConvertToHexAscii 344
 CStringHelper:: EAllowLws 351
 CStringHelper:: EAllowLws enumeration 351
 CStringHelper:: EscapeChar 344
 CStringHelper:: GetEscaped 345
 CStringHelper:: IsAlpha 345
 CStringHelper:: IsAlphaNum 345
 CStringHelper:: IsDigit 346
 CStringHelper:: IsFloat 346
 CStringHelper:: IsHexadecimal 346
 CStringHelper:: IsLineTerminator 347
 CStringHelper:: IsLWS 347
 CStringHelper:: IsNumeric 348
 CStringHelper:: IsStringQdTextOrQuotedPair 348
 CStringHelper:: ms_cCR 342
 CStringHelper:: ms_cLF 342
 CStringHelper:: ms_cNUL 342
 CStringHelper:: ms_cSP 343
 CStringHelper:: ms_cTAB 343
 CStringHelper:: ms_pszCRLF 343
 CStringHelper:: ms_pszDASHBOUNDARY 343
 CStringHelper:: ms_pszDOUBLECRLF 343
 CStringHelper:: QuotedStringToString 348
 CStringHelper:: RemoveVisualSeparators 349
 CStringHelper:: SkipLWS 349
 CStringHelper:: SkipToData 349
 CStringHelper:: SkipWSP 350
 CStringHelper:: StringToQuotedString 350
 CTelUri 351
 CTelUri class 351
 != 359
 ~CTelUri 354
 = 360
 == 360
 CTelUri 354

GenerateCopy 354	CToken 362, 363
GetParamList 355	FindTokenEnd 364
GetPhoneContext 355	Get CharSet 364
GetPhoneNumber 355	GetFloat 364
GetScheme 355	GetInt32 365
GetUriType 356	GetParserGrammar 365
IsEquivalent 356	GetString 366
IsGlobalPhoneNumber 356	GetUInt16 366
Parse 357	GetUInt32 365
Reset 358	GetUInt64 366
Serialize 358	IsAlpha 366
SetGlobalNumber 358	IsEmpty 367
SetLocalNumber 358	IsFloat 367
SetParamList 359	IsHexadecimal 367
CTelUri::!= 359	IsInt32 367
CTelUri::~CTelUri 354	IsLwsAllowed 368
CTelUri::= 360	IsNumeric 368
CTelUri::== 360	IsSignedInteger 369
CTelUri::CTelUri 354	IsUInt16 369
CTelUri::GenerateCopy 354	IsUInt32 370
CTelUri::GetParamList 355	IsUInt64 370
CTelUri::GetPhoneContext 355	IsUnsignedInteger 370, 371
CTelUri::GetPhoneNumber 355	IsValidChar 371
CTelUri::GetScheme 355	Length 371
CTelUri::GetUriType 356	Parse 372
CTelUri::IsEquivalent 356	Reset 372
CTelUri::IsGlobalPhoneNumber 356	Serialize 372
CTelUri::Parse 357	SetFloat 372
CTelUri::Reset 358	SetParserGrammar 373
CTelUri::Serialize 358	CToken::!= 373, 374
CTelUri::SetGlobalNumber 358	CToken::~CToken 363
CTelUri::SetLocalNumber 358	CToken::< 374
CTelUri::SetParamList 359	CToken::= 375, 376, 377
CToken 360	CToken::== 377
CToken class 360	CToken::CToken 362, 363
!= 373, 374	CToken::ECharSet 378
~CToken 363	CToken::ECharSet enumeration 378
< 374	CToken::FindTokenEnd 364
= 375, 376, 377	CToken::Get CharSet 364
== 377	CToken::GetFloat 364

CToken::GetInt32 365
 CToken::GetParserGrammar 365
 CToken::GetString 366
 CToken::GetUInt16 366
 CToken::GetUInt32 365
 CToken::GetUInt64 366
 CToken::IsAlpha 366
 CToken::IsEmpty 367
 CToken::IsFloat 367
 CToken::IsHexadecimal 367
 CToken::IsInt32 367
 CToken::IsLwsAllowed 368
 CToken::IsNumeric 368
 CToken::IsSignedInteger 369
 CToken::IsUInt16 369
 CToken::IsUInt32 370
 CToken::IsUInt64 370
 CToken::IsUnsignedInteger 370, 371
 CToken::IsValidChar 371
 CToken::Length 371
 CToken::Parse 372
 CToken::Reset 372
 CToken::Serialize 372
 CToken::SetFloat 372
 CToken::SetParserGrammar 373
 CUriFactory 378
 CUriFactory class 378

- CompareScheme 379
- ParseUri 379

 CUriFactory::CompareScheme 379
 CUriFactory::ParseUri 379

D

Draft-ietf-sip-outbound-15 Header parameter names 396
 draft-ietf-sip-publish-04.txt status codes 392
 draft-ietf-sip-referredby-05.txt status codes 392
 draft-ietf-sip-resource-priority-03.txt status codes 394
 draft-ietf-sip-session-timer-14.txt status codes 393
 draft-ietf-sip-session-timer-15 Header parameter names 398
 Draft-levy-sip-diversion-08 Header parameter names 395

E

eABSOLUTE enumeration member 384
 eACTIVE enumeration member 69, 504, 542
 eALLOW_LWS enumeration member 351
 eALLOW_SPECIAL_CHARS enumeration member 383
 eASYNCHRONOUSLY_UPDATED enumeration member 90
 eAuth enumeration member 116
 eAuthInt enumeration member 116
 eBASIC_RESET enumeration member 297
 eBEHAVIOR_HANDLE_NO_PACKET enumeration member 99
 eBEHAVIOR_HANDLE_REQUESTS_AND_2XX_RESPONSES enumeration member 99
 eBEHAVIOR_HANDLE_REQUESTS_ONLY enumeration member 99
 eCLEAR_TARGET_HEADERS enumeration member 658
 eCOMMA_SEPARATED enumeration member 67
 eCOMPLETE_RESET enumeration member 297
 eCONFIG_IPV4 enumeration member 68
 eCONFIG_IPV4_AND_IPV6 enumeration member 68
 eCONFIG_IPV6 enumeration member 68
 eCopyRecordRoute enumeration member 458
 eCREATE_CONNECTION_ASYNCHRONOUSLY enumeration member 101
 eCRITICAL enumeration member 512
 eCRLF_SEPARATED enumeration member 67
 eCS_LAST enumeration member 378
 eCS_SIP_HEADER enumeration member 146, 378
 eCS_SIPHEADER_PARAM enumeration member 378
 eCS_SIPURI_HEADER enumeration member 378
 eCS_SIPURI_PARAM enumeration member 146, 378
 eCS_SIPURI_PASSWORD enumeration member 378
 eCS_SIPURI_USER enumeration member 378
 eCS_TELURI_PARAM enumeration member 146, 378
 eCS_TELURI_TELEPHONE_SUBSCRIBER enumeration member 378
 eDEACTIVATED enumeration member 503
 eDISALLOW_SPECIAL_CHARS enumeration member 383
 eDO_NOT_CREATE_CONNECTION enumeration member 101
 eDoNotCopyRecordRoute enumeration member 458
 eEXPLICIT enumeration member 504, 603

eFAIL_ON_UNKNOWN_QOP enumeration member 478
eFAILOVER_DEFAULT enumeration member 68
eFAILOVER_NO_TCP_ASSUMPTION enumeration member 68
eFORCE_IP_ADDRESS enumeration member 67
eGIVEUP enumeration member 503
eHDR_ACCEPT enumeration member 384
eHDR_ACCEPT_CONTACT enumeration member 384
eHDR_ACCEPT_ENCODING enumeration member 384
eHDR_ACCEPT_LANGUAGE enumeration member 384
eHDR_ACCEPT_RESOURCE_PRIORITY enumeration member 384
eHDR_ALERT_INFO enumeration member 384
eHDR_ALLOW enumeration member 384
eHDR_ALLOW_EVENTS enumeration member 384
eHDR_AUTHENTICATION_INFO enumeration member 384
eHDR_AUTHORIZATION enumeration member 384
eHDR_CALL_ID enumeration member 384
eHDR_CALL_INFO enumeration member 384
eHDR_CONTACT enumeration member 384
eHDR_CONTENT_DESCRIPTION enumeration member 384
eHDR_CONTENT_DISPOSITION enumeration member 384
eHDR_CONTENT_ENCODING enumeration member 384
eHDR_CONTENT_ID enumeration member 384
eHDR_CONTENT_LANGUAGE enumeration member 384
eHDR_CONTENT_LENGTH enumeration member 384
eHDR_CONTENT_TRANSFER_ENCODING enumeration member 384
eHDR_CONTENT_TYPE enumeration member 384
eHDR_CSEQ enumeration member 384
eHDR_DATE enumeration member 384
eHDR_DIVERSION enumeration member 384
eHDR_ERROR_INFO enumeration member 384
eHDR_EVENT enumeration member 384
eHDR_EXPIRES enumeration member 384
eHDR_EXTENSION enumeration member 384
eHDR_FLOW_TIMER enumeration member 384
eHDR_FROM enumeration member 384
eHDR_HISTORY_INFO enumeration member 384
eHDR_IDENTITY enumeration member 384
eHDR_IDENTITY_INFO enumeration member 384
eHDR_IN_REPLY_TO enumeration member 384
eHDR_JOIN enumeration member 384
eHDR_MAX_FORWARDS enumeration member 384
eHDR_MIME_VERSION enumeration member 384
eHDR_MIN_EXPIRES enumeration member 384
eHDR_MIN_SE enumeration member 384
eHDR_ORGANIZATION enumeration member 384
eHDR_P_ACCESS_NETWORK_INFO enumeration member 384
eHDR_P_ASSERTED_IDENTITY enumeration member 384
eHDR_P_ASSOCIATED_URI enumeration member 384
eHDR_P_CALLED_PARTY_ID enumeration member 384
eHDR_P_CHARGING_FUNCTION_ADDRESSES enumeration member 384
eHDR_P_CHARGING_VECTOR enumeration member 384
eHDR_P_MEDIA_AUTHORIZATION enumeration member 384
eHDR_P_PREFERRED_IDENTITY enumeration member 384
eHDR_P_VISITED_NETWORK_ID enumeration member 384
eHDR_PATH enumeration member 384
eHDR_PRIORITY enumeration member 384
eHDR_PRIVACY enumeration member 384
eHDR_PROXY_AUTHENTICATE enumeration member 384
eHDR_PROXY_AUTHORIZATION enumeration member 384
eHDR_PROXY_REQUIRE enumeration member 384
eHDR_RACK enumeration member 384
eHDR_REASON enumeration member 384
eHDR_RECORD_ROUTE enumeration member 384
eHDR_REFER_TO enumeration member 384
eHDRREFERRED_BY enumeration member 384
eHDR_REJECT_CONTACT enumeration member 384
eHDR_REPLACE enumeration member 384
eHDR_REPLY_TO enumeration member 384
eHDR_REQUEST_DISPOSITION enumeration member 384
eHDR_REQUIRE enumeration member 384
eHDR_RESOURCE_PRIORITY enumeration member 384
eHDR_RETRY_AFTER enumeration member 384
eHDR_ROUTE enumeration member 384
eHDR_RSEQ enumeration member 384
eHDR_SERVER enumeration member 384
eHDR_SERVICE_ROUTE enumeration member 384
eHDR_SESSION_EXPIRES enumeration member 384

eHDR_SIP_ETAG enumeration member 384
eHDR_SIP_IF_MATCH enumeration member 384
eHDR SUBJECT enumeration member 384
eHDR_SUBSCRIPTION_STATE enumeration member 384
eHDR_SUPPORTED enumeration member 384
eHDR_TARGET_DIALOG enumeration member 384
eHDR_TIMESTAMP enumeration member 384
eHDR_TO enumeration member 384
eHDR_UNSUPPORTED enumeration member 384
eHDR_USER_AGENT enumeration member 384
eHDR_VIA enumeration member 384
eHDR_WARNING enumeration member 384
eHDR_WWW_AUTHENTICATE enumeration member 384
eHEADER enumeration member 512
eHIGH_PRIORITY_OBSERVER enumeration member 469
eID enumeration member 512
eID_PARAM_ABSENT_FOR_FIRST_REFERER enumeration member 535
eID_PARAM_ALWAYS_PRESENT enumeration member 535
eIGNORE_UNKNOWN_QOP enumeration member 478
eIM enumeration member 384
eIMPLICIT enumeration member 504, 603
eIMPLICIT_NOEXPIRES enumeration member 504
eIN_DIALOG enumeration member 658
eINACTIVE enumeration member 69
eINVALID enumeration member 512
eLOCAL enumeration member 469, 590
eLONG_FORM_NAME enumeration member 297
eLOOPBACK enumeration member 460
eLOW_PRIORITY_OBSERVER enumeration member 469
eMAILBOX enumeration member 384
eMANDATORY_ANGLE_BRACKET enumeration member 210
eMATCH_HOST_PORT enumeration member 558
eMATCH_URI enumeration member 558
eMATCH_USERNAME enumeration member 558
eMD5 enumeration member 116
eMD5_Session enumeration member 116
eNEXT_HDRS_RESET enumeration member 297
eNO_LWS enumeration member 351
eNO_REASON enumeration member 503

eNO_RESOURCE enumeration member 503
eNO_SEP enumeration member 154
eNone enumeration member 116
eNONE enumeration member 96, 512, 590
Enumerations 77, 126, 384
eOPTIONAL_ANGLE_BRACKET enumeration member 210
eOUT_OF_DIALOG enumeration member 658
ePARAM_CREATE_NEW enumeration member 297, 341
ePARAM_DONT_CREATE enumeration member 297, 341
ePENDING enumeration member 504, 542
ePREFER_FQDN enumeration member 67
ePREFER_LONG enumeration member 68
ePREFER_SHORT enumeration member 68
ePRES enumeration member 384
ePRESERVE_TARGET_HEADERS enumeration member 658
ePROBATION enumeration member 503
eQopMask enumeration member 116
eRAW_HDR_RESET enumeration member 297
eRECEIVED enumeration member 460, 658
eRefreshSession enumeration member 589
eREJECTED enumeration member 503
eREMOTE enumeration member 469, 590
eREMOVE_ROUTE enumeration member 96
eSECURE enumeration member 341
eSENT enumeration member 460, 658
eSESSION enumeration member 512
eSHORT_FORM_NAME enumeration member 297
eSIP enumeration member 384
eSIP_METHOD_ACK enumeration member 387
eSIP_METHOD_BYE enumeration member 387
eSIP_METHOD_CANCEL enumeration member 387
eSIP_METHOD_INFO enumeration member 387
eSIP_METHOD_INVITE enumeration member 387
eSIP_METHOD_MESSAGE enumeration member 387
eSIP_METHOD_NOTIFY enumeration member 387
eSIP_METHOD_OPTIONS enumeration member 387
eSIP_METHOD_PING enumeration member 387
eSIP_METHOD_PRACK enumeration member 387
eSIP_METHOD_PUBLISH enumeration member 387
eSIP_METHOD_REFER enumeration member 387

eSIP_METHOD_REGISTER enumeration member 387
 eSIP_METHOD_SERVICE enumeration member 387
 eSIP_METHOD_SUBSCRIBE enumeration member 387
 eSIP_METHOD_UNKNOWN enumeration member 387
 eSIP_METHOD_UPDATE enumeration member 387
 eSIP_STATUS_CLASS_CLIENT_ERROR enumeration member 388
 eSIP_STATUS_CLASS_GLOBAL_FAILURE enumeration member 388
 eSIP_STATUS_CLASS_INFORMATIONAL enumeration member 388
 eSIP_STATUS_CLASS_REDIRECTION enumeration member 388
 eSIP_STATUS_CLASS_SERVER_ERROR enumeration member 388
 eSIP_STATUS_CLASS_SUCCESS enumeration member 388
 eSIP_STATUS_CLASS_UNKNOWN enumeration member 388
 ESipHeaderType 384
 ESipHeaderType enumeration 384
 ESipMethod 387
 ESipMethod enumeration 387
 eSIPS enumeration member 384
 ESipStatusClass 388
 ESipStatusClass enumeration 388
 eSTATEFUL enumeration member 109, 124, 126
 eSTATELESS enumeration member 109, 124, 126
 eStopSessionTimer enumeration member 589
 eSYNCHRONOUSLY_UPDATED enumeration member 90
 eTEL enumeration member 384
 eTERMINATED enumeration member 542
 eTHRESHOLD_DYNAMIC enumeration member 558
 eTHRESHOLD_FIXED enumeration member 558
 eTIMEOUT enumeration member 503
 eUNSECURE enumeration member 341
 eUPDATE_TO_HEADER enumeration member 96
 eUSER enumeration member 512
 eWAITING_FOR_DIALOG_COMPLETION enumeration member 658
 eWITH_SEP enumeration member 154

F

friend class CSipPacketParser 311

Functions 388

G

g_aszMETHOD_NAME 391
 g_aszMETHOD_NAME variable 391
 g_auHeaderOrder 392
 g_auHeaderOrder variable 392
 g_stSipStackSipCore 2
 g_stSipStackSipCoreCSipContext 2
 g_stSipStackSipCoreCSipContext variable 2
 g_stSipStackSipCoreCSipContextFeatureECOM 2
 g_stSipStackSipCoreCSipContextFeatureECOM variable 2
 g_stSipStackSipCoreCSipCoreConfig 2
 g_stSipStackSipCoreCSipCoreConfig variable 2
 g_stSipStackSipCoreCSipCoreConfigFeatureECOM 2
 g_stSipStackSipCoreCSipCoreConfigFeatureECOM variable 2
 g_stSipStackSipCoreCSipCoreEventList 2
 g_stSipStackSipCoreCSipCoreEventList variable 2
 g_stSipStackSipCoreCSipDialogMatcherList 2
 g_stSipStackSipCoreCSipDialogMatcherList variable 2
 g_stSipStackSipCoreCSipEntity 2
 g_stSipStackSipCoreCSipEntity variable 2
 g_stSipStackSipCoreCSipForkedDialogGrouper 2
 g_stSipStackSipCoreCSipForkedDialogGrouper variable 2
 g_stSipStackSipCoreCSipNetworkInterfaceList 2
 g_stSipStackSipCoreCSipNetworkInterfaceList variable 2
 g_stSipStackSipCoreCSipReqCtxConnectionSvc 2
 g_stSipStackSipCoreCSipReqCtxConnectionSvc variable 2
 g_stSipStackSipCoreCSipReqCtxCoreSvc 2
 g_stSipStackSipCoreCSipReqCtxCoreSvc variable 2
 g_stSipStackSipCoreCSipRequestContext 2
 g_stSipStackSipCoreCSipRequestContext variable 2
 g_stSipStackSipCoreCSipRequestContextFeatureECOM 2
 g_stSipStackSipCoreCSipRequestContextFeatureECOM variable 2
 g_stSipStackSipCoreCSipStackMonitor 2
 g_stSipStackSipCoreCSipStackMonitor variable 2
 g_stSipStackSipCoreResultIdSipCore 2
 g_stSipStackSipCoreResultIdSipCore variable 2
 g_stSipStackSipCoreSvc 2
 g_stSipStackSipCoreSvcCReqCtxSipDestinationSelectionSvc 2

g_stSipStackSipCoreSvcCReqCtxSipDestinationSelectionSvc
 variable 2
 g_stSipStackSipCoreSvcCServerLocator 2
 g_stSipStackSipCoreSvcCServerLocator variable 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklist 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklist variable 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistFeatureECOM 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistFeatureECOM
 variable 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistSvc 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistSvc variable 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistSvcFeatureECOM
 2
 g_stSipStackSipCoreSvcCSipConnectionBlacklistSvcFeatureECOM
 variable 2
 g_stSipStackSipCoreSvcCSipConnectionReuseReqCtxCoreSvc 2
 g_stSipStackSipCoreSvcCSipConnectionReuseReqCtxCoreSvc
 variable 2
 g_stSipStackSipCoreSvcCSipConnectionReuseSvc 2
 g_stSipStackSipCoreSvcCSipConnectionReuseSvc variable 2
 g_stSipStackSipCoreSvcCSipCoreOutputControllingSvc 2
 g_stSipStackSipCoreSvcCSipCoreOutputControllingSvc variable 2
 g_stSipStackSipCoreSvcCSipCoreOutputControllingSvcFeatureEC
 OM
 2
 g_stSipStackSipCoreSvcCSipCoreOutputControllingSvcFeatureEC
 OM
 variable 2
 g_stSipStackSipCoreSvcCSipDestinationSelectionSvc 2
 g_stSipStackSipCoreSvcCSipDestinationSelectionSvc variable 2
 g_stSipStackSipCoreSvcCSipEnumRequestHandler 2
 g_stSipStackSipCoreSvcCSipEnumRequestHandler variable 2
 g_stSipStackSipCoreSvcCSipEnumSvc 2
 g_stSipStackSipCoreSvcCSipEnumSvc variable 2
 g_stSipStackSipCoreSvcCSipEnumSvcFeatureECOM 2
 g_stSipStackSipCoreSvcCSipEnumSvcFeatureECOM variable 2
 g_stSipStackSipCoreSvcCSipOutboundReqCtxSvc 2
 g_stSipStackSipCoreSvcCSipOutboundReqCtxSvc variable 2
 g_stSipStackSipCoreSvcCSipOutboundSvc 2
 g_stSipStackSipCoreSvcCSipOutboundSvc variable 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionList 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionList variable 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionSvc 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionSvc variable 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionSvcFeatureECO
 M
 2
 g_stSipStackSipCoreSvcCSipPersistentConnectionSvcFeatureECO
 M
 variable 2
 g_stSipStackSipCoreSvcCSipReqCtxServerLocationSvc 2
 g_stSipStackSipCoreSvcCSipReqCtxServerLocationSvc variable 2
 g_stSipStackSipCoreSvcCSipServerLocationSvc 2
 g_stSipStackSipCoreSvcCSipServerLocationSvc variable 2
 g_stSipStackSipCoreSvcCSipServerLocationSvcFeatureECOM 2
 g_stSipStackSipCoreSvcCSipServerLocationSvcFeatureECOM
 variable 2
 g_stSipStackSipCoreSvcCSipServerMonitor 2
 g_stSipStackSipCoreSvcCSipServerMonitor variable 2
 g_stSipStackSipCoreSvcCSipServerMonitorFeatureECOM 2
 g_stSipStackSipCoreSvcCSipServerMonitorFeatureECOM variable 2
 g_stSipStackSipCoreSvcCSipServerMonitorSvc 2
 g_stSipStackSipCoreSvcCSipServerMonitorSvc variable 2
 g_stSipStackSipCoreSvcCSipServerMonitorSvcFeatureECOM 2
 g_stSipStackSipCoreSvcCSipServerMonitorSvcFeatureECOM
 variable 2
 g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvc 2
 g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvc
 variable 2
 g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvcFeatur
 eECOM
 2
 g_stSipStackSipCoreSvcCSipStatelessDigestServerAuthSvcFeatur
 eECOM
 variable 2
 g_stSipStackSipCoreSvcCSipStatisticsContainer 2
 g_stSipStackSipCoreSvcCSipStatisticsContainer variable 2
 g_stSipStackSipCoreSvcCSipStatisticsReqCtxSvc 2
 g_stSipStackSipCoreSvcCSipStatisticsReqCtxSvc variable 2
 g_stSipStackSipCoreSvcCSipStatisticsSvc 2
 g_stSipStackSipCoreSvcCSipStatisticsSvc variable 2
 g_stSipStackSipCoreSvcCSipStatisticsSvcFeatureECOM 2
 g_stSipStackSipCoreSvcCSipStatisticsSvcFeatureECOM variable 2
 g_stSipStackSipCoreSvcCSipSymmetricUdpSvc 2
 g_stSipStackSipCoreSvcCSipSymmetricUdpSvc variable 2

g_stSipStackSipCoreSvcCSipSymmetricUdpSvcFeatureECOM 2
 g_stSipStackSipCoreSvcCSipSymmetricUdpSvcFeatureECOM
 variable 2
 g_stSipStackSipCoreSvcCSipViaManagementSvc 2
 g_stSipStackSipCoreSvcCSipViaManagementSvc variable 2
 g_stSipStackSipCoreSvcResultIdSipCoreSvc 2
 g_stSipStackSipCoreSvcResultIdSipCoreSvc variable 2
 g_stSipStackSipParser 2
 g_stSipStackSipParserCDate 2
 g_stSipStackSipParserCDate variable 2
 g_stSipStackSipParserCSipHeader 2
 g_stSipStackSipParserCSipHeader variable 2
 g_stSipStackSipParserResultIdSipParser 2
 g_stSipStackSipParserResultIdSipParser variable 2
 g_stSipStackSipProxy 3
 g_stSipStackSipProxyCSipProxyConfig 3
 g_stSipStackSipProxyCSipProxyConfig variable 3
 g_stSipStackSipProxyCSipProxyHelper 3
 g_stSipStackSipProxyCSipProxyHelper variable 3
 g_stSipStackSipProxyCSipSessionStatefulProxySvc 3
 g_stSipStackSipProxyCSipSessionStatefulProxySvc variable 3
 g_stSipStackSipProxyCSipSessionStatefulProxySvcFeatureECOM
 3
 g_stSipStackSipProxyCSipSessionStatefulProxySvcFeatureECOM
 variable 3
 g_stSipStackSipProxyCSipStatelessProxySvc 3
 g_stSipStackSipProxyCSipStatelessProxySvc variable 3
 g_stSipStackSipProxyCSipStatelessProxySvcFeatureECOM 3
 g_stSipStackSipProxyCSipStatelessProxySvcFeatureECOM
 variable 3
 g_stSipStackSipProxyCSipTransactionStatefulProxySvc 3
 g_stSipStackSipProxyCSipTransactionStatefulProxySvc variable 3
 g_stSipStackSipProxyCSipTransactionStatefulProxySvcFeatureEC
 OM
 3
 g_stSipStackSipProxyCSipTransactionStatefulProxySvcFeatureEC
 OM
 variable 3
 g_stSipStackSipTransaction 3
 g_stSipStackSipTransactionCSipClientInviteTransaction 3
 g_stSipStackSipTransactionCSipClientInviteTransaction variable 3
 g_stSipStackSipTransactionCSipClientNonInviteTransaction 3
 g_stSipStackSipTransactionCSipClientNonInviteTransaction
 variable 3
 g_stSipStackSipTransactionCSipServerInviteTransaction 3
 g_stSipStackSipTransactionCSipServerInviteTransaction variable 3
 g_stSipStackSipTransactionCSipServerNonInviteTransaction 3
 g_stSipStackSipTransactionCSipServerNonInviteTransaction
 variable 3
 g_stSipStackSipTransactionCSipTransaction 3
 g_stSipStackSipTransactionCSipTransaction variable 3
 g_stSipStackSipTransactionCSipTransactionMgr 3
 g_stSipStackSipTransactionCSipTransactionMgr variable 3
 g_stSipStackSipTransactionResultIdSipTransaction 3
 g_stSipStackSipTransactionResultIdSipTransaction variable 3
 g_stSipStackSipTransport 3
 g_stSipStackSipTransportCSipAsyncSocketFactoryConfigurationMg
 3
 g_stSipStackSipTransportCSipAsyncSocketFactoryConfigurationMg
 variable 3
 g_stSipStackSipTransportCSipClientSocket 3
 g_stSipStackSipTransportCSipClientSocket variable 3
 g_stSipStackSipTransportCSipConnectionSvc 3
 g_stSipStackSipTransportCSipConnectionSvc variable 3
 g_stSipStackSipTransportCSipDefaultDataLogger 3
 g_stSipStackSipTransportCSipDefaultDataLogger variable 3
 g_stSipStackSipTransportCSipPacket 3
 g_stSipStackSipTransportCSipPacket variable 3
 g_stSipStackSipTransportCSipParserSvc 3
 g_stSipStackSipTransportCSipParserSvc variable 3
 g_stSipStackSipTransportCSipServerSocket 3
 g_stSipStackSipTransportCSipServerSocket variable 3
 g_stSipStackSipTransportCSipSpirallingSvc 3
 g_stSipStackSipTransportCSipSpirallingSvc variable 3
 g_stSipStackSipTransportCSipTlsContextFactory 3
 g_stSipStackSipTransportCSipTlsContextFactory variable 3
 g_stSipStackSipTransportCSipTransportMgr 3
 g_stSipStackSipTransportCSipTransportMgr variable 3
 g_stSipStackSipTransportCSipTransportSvc 3
 g_stSipStackSipTransportCSipTransportSvc variable 3
 g_stSipStackSipTransportCSipTransportTools 3
 g_stSipStackSipTransportCSipTransportTools variable 3
 g_stSipStackSipTransportCSocketFactory 3

g_stSipStackSipTransportCSocketFactory variable 3
 g_stSipStackSipTransportResultIdSipTransport 3
 g_stSipStackSipTransportResultIdSipTransport variable 3
 g_stSipStackSipUserAgent 3
 g_stSipStackSipUserAgentCSipAutomaticAnswererReqCtxSvc 3
 g_stSipStackSipUserAgentCSipAutomaticAnswererReqCtxSvc variable 3
 g_stSipStackSipUserAgentCSipClientSvc 3
 g_stSipStackSipUserAgentCSipClientSvc variable 3
 g_stSipStackSipUserAgentCSipClientSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipClientSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipContextTerminator 3
 g_stSipStackSipUserAgentCSipContextTerminator variable 3
 g_stSipStackSipUserAgentCSipDigestClientAuthSvc 3
 g_stSipStackSipUserAgentCSipDigestClientAuthSvc variable 3
 g_stSipStackSipUserAgentCSipDigestClientAuthSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipDigestClientAuthSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipDiversionSvc 3
 g_stSipStackSipUserAgentCSipDiversionSvc variable 3
 g_stSipStackSipUserAgentCSipDiversionSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipDiversionSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvc 3
 g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvc variable 3
 g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipGenericReqCtxCoreSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipGenericSvc 3
 g_stSipStackSipUserAgentCSipGenericSvc variable 3
 g_stSipStackSipUserAgentCSipGenericSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipGenericSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipGlareSvc 3
 g_stSipStackSipUserAgentCSipGlareSvc variable 3
 g_stSipStackSipUserAgentCSipGlareSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipGlareSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipMwiSvc 3
 g_stSipStackSipUserAgentCSipMwiSvc variable 3
 g_stSipStackSipUserAgentCSipMwiSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipMwiSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipNotifierSvc 3
 g_stSipStackSipUserAgentCSipNotifierSvc variable 3
 g_stSipStackSipUserAgentCSipNotifierSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipNotifierSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipOptionTagsSvc 3
 g_stSipStackSipUserAgentCSipOptionTagsSvc variable 3
 g_stSipStackSipUserAgentCSipOptionTagsSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipOptionTagsSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipPrivacySvc 3
 g_stSipStackSipUserAgentCSipPrivacySvc variable 3
 g_stSipStackSipUserAgentCSipPrivacySvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipPrivacySvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipPublishSvc 3
 g_stSipStackSipUserAgentCSipPublishSvc variable 3
 g_stSipStackSipUserAgentCSipPublishSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipPublishSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipRedirectionSvc 3
 g_stSipStackSipUserAgentCSipRedirectionSvc variable 3
 g_stSipStackSipUserAgentCSipRedirectionSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipRedirectionSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipRefereeSvc 3
 g_stSipStackSipUserAgentCSipRefereeSvc variable 3
 g_stSipStackSipUserAgentCSipRefereeSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipRefereeSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipReferrerSvc 3
 g_stSipStackSipUserAgentCSipReferrerSvc variable 3
 g_stSipStackSipUserAgentCSipReferrerSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipReferrerSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipRegistrationSvc 3
 g_stSipStackSipUserAgentCSipRegistrationSvc variable 3
 g_stSipStackSipUserAgentCSipRegistrationSvcFeatureECOM 3
 g_stSipStackSipUserAgentCSipRegistrationSvcFeatureECOM variable 3
 g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvc 3
 g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvc variable 3

g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvcFeatureECOM 3	g_stSipStackSipUserAgentCSipSessionTransactionUasByeFeatureECOM variable 3
g_stSipStackSipUserAgentCSipReliableProvisionalResponseSvcFeatureECOM variable 3	g_stSipStackSipUserAgentCSipSessionTransactionUasInvite 3 g_stSipStackSipUserAgentCSipSessionTransactionUasInvite variable 3
g_stSipStackSipUserAgentCSipReplacesSvc 3	g_stSipStackSipUserAgentCSipSessionTransactionUasInviteFeatureECOM 3
g_stSipStackSipUserAgentCSipReplacesSvc variable 3	g_stSipStackSipUserAgentCSipSessionTransactionUasInviteFeatureECOM variable 3
g_stSipStackSipUserAgentCSipReplacesSvcFeatureECOM 3	g_stSipStackSipUserAgentCSipSessionTransactionUasInviteFeatureECOM variable 3
g_stSipStackSipUserAgentCSipReplacesSvcFeatureECOM variable 3	g_stSipStackSipUserAgentCSipSessionTransactionUasInviteFeatureECOM variable 3
g_stSipStackSipUserAgentCSipSessionSvc 3	g_stSipStackSipUserAgentCSipSubscriberSvc 3
g_stSipStackSipUserAgentCSipSessionSvc variable 3	g_stSipStackSipUserAgentCSipSubscriberSvc variable 3
g_stSipStackSipUserAgentCSipSessionSvcFeatureECOM 3	g_stSipStackSipUserAgentCSipSubscriberSvcFeatureECOM 3
g_stSipStackSipUserAgentCSipSessionSvcFeatureECOM variable 3	g_stSipStackSipUserAgentCSipSubscriberSvcFeatureECOM variable 3
g_stSipStackSipUserAgentCSipSessionTimerSvc 3	g_stSipStackSipUserAgentCSipTransactionCompletionReqCtxCon Svc 3
g_stSipStackSipUserAgentCSipSessionTimerSvc variable 3	g_stSipStackSipUserAgentCSipTransactionCompletionReqCtxCon Svc variable 3
g_stSipStackSipUserAgentCSipSessionTimerSvcFeatureECOM 3	g_stSipStackSipUserAgentCSipTransactionCompletionReqCtxCon Svc variable 3
g_stSipStackSipUserAgentCSipSessionTimerSvcFeatureECOM variable 3	g_stSipStackSipUserAgentCSipTransactionCompletionReqCtxCon Svc variable 3
g_stSipStackSipUserAgentCSipSessionTransaction 3	g_stSipStackSipUserAgentCSipTransactionCompletionSvc 3
g_stSipStackSipUserAgentCSipSessionTransaction variable 3	g_stSipStackSipUserAgentCSipTransactionCompletionSvc variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUacBye 3	g_stSipStackSipUserAgentCSipTransactionCompletionSvcFeatureE COM 3
g_stSipStackSipUserAgentCSipSessionTransactionUacBye variable 3	g_stSipStackSipUserAgentCSipTransactionCompletionSvcFeatureE COM variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUacByeFeature ECOM 3	g_stSipStackSipUserAgentCSipTransferSvc07 3
g_stSipStackSipUserAgentCSipSessionTransactionUacByeFeature ECOM variable 3	g_stSipStackSipUserAgentCSipTransferSvc07 variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUacInvite 3	g_stSipStackSipUserAgentCSipTransferSvc07FeatureECOM 3
g_stSipStackSipUserAgentCSipSessionTransactionUacInvite variable 3	g_stSipStackSipUserAgentCSipTransferSvc07FeatureECOM variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUacInviteFeatur eECOM 3	g_stSipStackSipUserAgentCSipUaAssertedIdentitySvc 3
g_stSipStackSipUserAgentCSipSessionTransactionUacInviteFeatur eECOM variable 3	g_stSipStackSipUserAgentCSipUaAssertedIdentitySvc variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUasBye 3	g_stSipStackSipUserAgentCSipUaAssertedIdentitySvcFeatureECO M 3
g_stSipStackSipUserAgentCSipSessionTransactionUasBye variable 3	g_stSipStackSipUserAgentCSipUaAssertedIdentitySvcFeatureECO M variable 3
g_stSipStackSipUserAgentCSipSessionTransactionUasByeFeature ECOM 3	g_stSipStackSipUserAgentCSipUaHelpers 3
	g_stSipStackSipUserAgentCSipUaHelpers variable 3
	g_stSipStackSipUserAgentCSipUpdateSvc 3

g_stSipStackSipUserAgentCSipUpdateSvc variable 3	ISipConnectionBlacklistMgr::EvAddressBlacklisted 84
g_stSipStackSipUserAgentCSipUpdateSvcFeatureECOM 3	ISipConnectionBlacklistMgr::EvBlacklistDurationCompleted 84
g_stSipStackSipUserAgentCSipUpdateSvcFeatureECOM variable 3	ISipConnectionBlacklistSvc 84
g_stSipStackSipUserAgentCSipUserAgentSvc 3	ISipConnectionBlacklistSvc class 84
g_stSipStackSipUserAgentCSipUserAgentSvc variable 3	ISipConnectionReuseSvc 85
g_stSipStackSipUserAgentCSipUserAgentSvcFeatureECOM 3	ISipConnectionReuseSvc class 85
g_stSipStackSipUserAgentCSipUserAgentSvcFeatureECOM variable 3	GetLocalAddress 86
g_stSipStackSipUserAgentResultIdSipUserAgent 3	GetPeerAddress 86
g_stSipStackSipUserAgentResultIdSipUserAgent variable 3	GetTransport 86
	SetLocalAddress 87
	SetPeerAddress 87
	SetTransport 87
	ISipConnectionReuseSvc::GetLocalAddress 86
	ISipConnectionReuseSvc::GetPeerAddress 86
	ISipConnectionReuseSvc::GetTransport 86
	ISipConnectionReuseSvc::SetLocalAddress 87
	ISipConnectionReuseSvc::SetPeerAddress 87
	ISipConnectionReuseSvc::SetTransport 87
	ISipContext 23
	ISipContext class 23
	AttachService 25
	Clear 26
	GetEntityId 26
	GetOpaque 26
	OnPacketReceived 27
	SetEntityId 27
	SetOpaque 27
	ISipContext::AttachService 25
	ISipContext::Clear 26
	ISipContext::GetEntityId 26
	ISipContext::GetOpaque 26
	ISipContext::OnPacketReceived 27
	ISipContext::SetEntityId 27
	ISipContext::SetOpaque 27
	ISipCoreConfig 28
	ISipCoreConfig class 28
	AddLocalAddress 33
	AddServerToResponses 34
	AddStackVersionTold 35
	AddSupportedExtension 35
Header Parameters 395	
Introduction 1	
ISipClientEventControl 19	
ISipClientEventControl class 19	
CallNextClientEvent 19	
ClearClientEvents 20	
GetOpaque 20	
RelissueRequest 20	
SetOpaque 21	
ISipClientEventControl::CallNextClientEvent 19	
ISipClientEventControl::ClearClientEvents 20	
ISipClientEventControl::GetOpaque 20	
ISipClientEventControl::RelissueRequest 20	
ISipClientEventControl::SetOpaque 21	
ISipClientTransaction 21	
ISipClientTransaction class 21	
CancelRequest 22	
GetOpaque 23	
SetOpaque 23	
ISipClientTransaction::CancelRequest 22	
ISipClientTransaction::GetOpaque 23	
ISipClientTransaction::SetOpaque 23	
ISipConnectionBlacklistMgr 83	
ISipConnectionBlacklistMgr class 83	
EvAddressBlacklisted 84	
EvBlacklistDurationCompleted 84	

AddTransportObserverA	35	SetPrincipalBufferSize	55
AddUserAgentToRequests	36	SetSipDataLogger	55
AddViaRportParam	36	SetSocketClosureType	56
CloseAllConnections	37	SetSpirallingSvcState	56
EnableOutboundCrlfKeepAliveA	37	SetSupportedDnsQueries	56
ForceVisibleLocalAddress	37	SetSupportedIpVersion	57
GetConnectionBlacklistInstance	38	SetSupportedSipTransport	58
GetNetworkInterfaceList	39	SetT1	58
GetPersistentConnectionList	39	SetT2	59
GetSupportedExtensions	39	SetT4	59
GetSupportedIpVersion	40	SetTimeoutTimer	59
GetTlsContextFactory	40	SetTimerB	60
GetUaHelpers	40	SetTimerD	61
GetVersion	41	SetTimerF	61
ListenA	41	SetTimerH	62
RemoveLocalAddress	41	SetTimerJ	62
RemoveSupportedExtension	42	SetTlsSessionCacheMaxSize	63
RemoveTransportObserverA	42	SetTransportThread	63
SetApplicationId	43	SetUaResponseMultipleViasCheck	64
SetClientAuthUnknownQopBehavior	43	SetUdpMaxSizeThreshold	65
SetCommaSeparatedHeader	43	SetViaAddressType	65
SetConnectionBlacklistInstance	44	ShutdownA	65
SetConnectionParameters	44	Startup	66
SetCoreThread	46	StopListeningA	66
SetCoreUser	47	UpdateParserGrammar	67
SetCSeq64BitsSupport	47	ISipCoreConfig::AddLocalAddress	33
SetDefaultDialogMatcherList	47	ISipCoreConfig::AddServerToResponses	34
SetDnsResolverThread	48	ISipCoreConfig::AddStackVersionTold	35
SetEntityId	48	ISipCoreConfig::AddSupportedExtension	35
SetEnumE164ZoneSuffix	49	ISipCoreConfig::AddTransportObserverA	35
SetFailoverMode	49	ISipCoreConfig::AddUserAgentToRequests	36
SetHandshakeValidatorCallback	50	ISipCoreConfig::AddViaRportParam	36
SetHeaderFormPreference	51	ISipCoreConfig::CloseAllConnections	37
SetKeepAliveExtensionMgrA	52	ISipCoreConfig::EAddressTypePreference	67
SetMaxForwards	52	ISipCoreConfig::EAddressTypePreference enumeration	67
SetMaxPayloadSize	52	ISipCoreConfig::ECommaSeparatedHeaderConfig	67
SetMaxReceivePacketSize	53	ISipCoreConfig::ECommaSeparatedHeaderConfig enumeration	67
SetMaxSendBufferSize	53	ISipCoreConfig::EFailoverMode	68
SetPacketInspectorCallback	54	ISipCoreConfig::EFailoverMode enumeration	68
SetPacketModifierCallback	54	ISipCoreConfig::EHeaderFormPreference	68

ISipCoreConfig::EHeaderFormPreference enumeration 68	ISipCoreConfig::SetMaxSendBufferSize 53
ISipCoreConfig::EIpVersionConfig 68	ISipCoreConfig::SetPacketInspectorCallback 54
ISipCoreConfig::EIpVersionConfig enumeration 68	ISipCoreConfig::SetPacketModifierCallback 54
ISipCoreConfig::EnableOutboundCrlfKeepAliveA 37	ISipCoreConfig::SetPrincipalBufferSize 55
ISipCoreConfig::ESpirallingSvcState 69	ISipCoreConfig::SetSipDataLogger 55
ISipCoreConfig::ESpirallingSvcState enumeration 69	ISipCoreConfig::SetSocketClosureType 56
ISipCoreConfig::ForceVisibleLocalAddress 37	ISipCoreConfig::SetSpirallingSvcState 56
ISipCoreConfig::GetConnectionBlacklistInstance 38	ISipCoreConfig::SetSupportedDnsQueries 56
ISipCoreConfig::GetNetworkInterfaceList 39	ISipCoreConfig::SetSupportedIpVersion 57
ISipCoreConfig::GetPersistentConnectionList 39	ISipCoreConfig::SetSupportedSipTransport 58
ISipCoreConfig::GetSupportedExtensions 39	ISipCoreConfig::SetT1 58
ISipCoreConfig::GetSupportedIpVersion 40	ISipCoreConfig::SetT2 59
ISipCoreConfig::GetTlsContextFactory 40	ISipCoreConfig::SetT4 59
ISipCoreConfig::GetUaHelpers 40	ISipCoreConfig::SetTimeoutTimer 59
ISipCoreConfig::GetVersion 41	ISipCoreConfig::SetTimerB 60
ISipCoreConfig::ListenA 41	ISipCoreConfig::SetTimerD 61
ISipCoreConfig::RemoveLocalAddress 41	ISipCoreConfig::SetTimerF 61
ISipCoreConfig::RemoveSupportedExtension 42	ISipCoreConfig::SetTimerH 62
ISipCoreConfig::RemoveTransportObserverA 42	ISipCoreConfig::SetTimerJ 62
ISipCoreConfig::SAccessibleNetwork 31	ISipCoreConfig::SetTlsSessionCacheMaxSize 63
ISipCoreConfig::SAccessibleNetwork struct 31	ISipCoreConfig::SetTransportThread 63
ISipCoreConfig::SetApplicationId 43	ISipCoreConfig::SetUaResponseMultipleViasCheck 64
ISipCoreConfig::SetClientAuthUnknownQopBehavior 43	ISipCoreConfig::SetUdpMaxSizeThreshold 65
ISipCoreConfig::SetCommaSeparatedHeader 43	ISipCoreConfig::SetViaAddressType 65
ISipCoreConfig::SetConnectionBlacklistInstance 44	ISipCoreConfig::ShutdownA 65
ISipCoreConfig::SetConnectionParameters 44	ISipCoreConfig::SNetworkIf 32
ISipCoreConfig::SetCoreThread 46	ISipCoreConfig::SNetworkIf struct 32
ISipCoreConfig::SetCoreUser 47	m_socketAddr 33
ISipCoreConfig::SetCSeq64BitsSupport 47	m_uListeningTlsPort 33
ISipCoreConfig::SetDefaultDialogMatcherList 47	m_uListeningUdpTcpPort 33
ISipCoreConfig::SetDnsResolverThread 48	SNetworkIf 33
ISipCoreConfig::SetEntityId 48	ISipCoreConfig::SNetworkIf::m_socketAddr 33
ISipCoreConfig::SetEnumE164ZoneSuffix 49	ISipCoreConfig::SNetworkIf::m_uListeningTlsPort 33
ISipCoreConfig::SetFailoverMode 49	ISipCoreConfig::SNetworkIf::m_uListeningUdpTcpPort 33
ISipCoreConfig::SetHandshakeValidatorCallback 50	ISipCoreConfig::SNetworkIf::SNetworkIf 33
ISipCoreConfig::SetHeaderFormPreference 51	ISipCoreConfig::Startup 66
ISipCoreConfig::SetKeepAliveExtensionMgrA 52	ISipCoreConfig::StopListeningA 66
ISipCoreConfig::SetMaxForwards 52	ISipCoreConfig::UpdateParserGrammar 67
ISipCoreConfig::SetMaxPayloadSize 52	ISipCoreOutputControllingMgr 88
ISipCoreConfig::SetMaxReceivePacketSize 53	ISipCoreOutputControllingMgr class 88

EvUpdatePacket 88	ISipDialogMatcher::SetDialogMatcherList 72
ISipCoreOutputControllingMgr::EUpdatingSynchronization 90	ISipDigestClientAuthMgr 472
ISipCoreOutputControllingMgr::EUpdatingSynchronization enumeration 90	ISipDigestClientAuthMgr class 472
ISipCoreOutputControllingMgr::EvUpdatePacket 88	EvAuthLoop 472
ISipCoreOutputControllingSvc 90	EvCredentialsExist 473
ISipCoreOutputControllingSvc class 90	EvCredentialsRequired 473
PacketAsynchronouslyUpdated 91	EvInvalidCredentials 474
SetManager 91	ISipDigestClientAuthMgr::EvAuthLoop 472
ISipCoreOutputControllingSvc::PacketAsynchronouslyUpdated 91	ISipDigestClientAuthMgr::EvCredentialsExist 473
ISipCoreOutputControllingSvc::SetManager 91	ISipDigestClientAuthMgr::EvCredentialsRequired 473
ISipCoreUser 69	ISipDigestClientAuthMgr::EvInvalidCredentials 474
ISipCoreUser class 69	ISipDigestClientAuthSvc 474
EvCommandResult 69	ISipDigestClientAuthSvc class 474
EvOnPacketReceived 70	Authenticate 476
EvShutdownCompleted 70	EnableQualityOfProtectionAuthInt 476
ISipCoreUser::EvCommandResult 69	GetChallenges 477
ISipCoreUser::EvOnPacketReceived 70	Reset 477
ISipCoreUser::EvShutdownCompleted 70	SetLoopThreshold 477
ISipDataLogger 458	SetManager 478
ISipDataLogger class 458	SetRouteUriAuthentication 478
~ISipDataLogger 459	ISipDigestClientAuthSvc::Authenticate 476
LogRawData 459	ISipDigestClientAuthSvc::EnableQualityOfProtectionAuthInt 476
LogSipPacket 460	ISipDigestClientAuthSvc::EUknownQopBehavior 478
ISipDataLogger::~ISipDataLogger 459	ISipDigestClientAuthSvc::EUknownQopBehavior enumeration 478
ISipDataLogger::EDirection 460	ISipDigestClientAuthSvc::GetChallenges 477
ISipDataLogger::EDirection enumeration 460	ISipDigestClientAuthSvc::Reset 477
ISipDataLogger::LogRawData 459	ISipDigestClientAuthSvc::SChallengeData 475
ISipDataLogger::LogSipPacket 460	ISipDigestClientAuthSvc::SChallengeData struct 475
ISipDestinationSelectionSvc 92	ISipDigestClientAuthSvc::SetLoopThreshold 477
ISipDestinationSelectionSvc class 92	ISipDigestClientAuthSvc::SetManager 478
ForceDestination 93	ISipDigestClientAuthSvc::SetRouteUriAuthentication 478
ISipDestinationSelectionSvc::ForceDestination 93	ISipDiversionSvc 93
ISipDialogMatcher 71	ISipDiversionSvc class 93
ISipDialogMatcher class 71	SetReason 94
GetDialogMatcherList 71	ISipDiversionSvc::SetReason 94
OnPacketReceived 71	ISipEnumRequestHandlerMgr 94
SetDialogMatcherList 72	ISipEnumRequestHandlerMgr class 94
ISipDialogMatcher::GetDialogMatcherList 71	OnGetEnumUriAResult 94
ISipDialogMatcher::OnPacketReceived 71	ISipEnumRequestHandlerMgr::OnGetEnumUriAResult 94
	ISipEnumSvc 95

ISipEnumSvc class 95	ISipGlareSvc::SetManager 485
SetServiceBehaviour 95	
ISipEnumSvc::EEnumServiceBehaviour 96	ISipMwiMgr 485
ISipEnumSvc::EEnumServiceBehaviour enumeration 96	ISipMwiMgr class 485
ISipEnumSvc::SetServiceBehaviour 95	EvExpired 486
ISipForkedDialogGrouper 73	EvExpiring 486
ISipForkedDialogGrouper class 73	EvFailure 486
AddRef 73	EvIntervalTooSmall 486
ContextCreated 73	EvInvalidNotify 487
ReleaseRef 74	EvNotified 488
ISipForkedDialogGrouper::AddRef 73	EvProgress 488
ISipForkedDialogGrouper::ContextCreated 73	EvShutdown 488
ISipForkedDialogGrouper::ReleaseRef 74	EvSuccess 489
ISipForkedDialogGrouperMgr 74	ISipMwiMgr::EvExpired 486
ISipForkedDialogGrouperMgr class 74	ISipMwiMgr::EvExpiring 486
EvNewDialogNeeded 75	ISipMwiMgr::EvFailure 486
ISipForkedDialogGrouperMgr::EvNewDialogNeeded 75	ISipMwiMgr::EvIntervalTooSmall 486
ISipGenericMgr 479	ISipMwiMgr::EvInvalidNotify 487
ISipGenericMgr class 479	ISipMwiMgr::EvNotified 488
EvFailure 479	ISipMwiMgr::EvProgress 488
EvProgress 480	ISipMwiMgr::EvShutdown 488
EvRequest 480	ISipMwiMgr::EvSuccess 489
EvSuccess 480	ISipMwiSvc 489
ISipGenericMgr::EvFailure 479	ISipMwiSvc class 489
ISipGenericMgr::EvProgress 480	SetExpiringThreshold 490
ISipGenericMgr::EvRequest 480	SetManager 490
ISipGenericMgr::EvSuccess 480	Subscribe 490
ISipGenericSvc 481	Unsubscribe 491
ISipGenericSvc class 481	ISipMwiSvc::SetExpiringThreshold 490
SendRequest 482	ISipMwiSvc::SetManager 490
SetManager 482	ISipMwiSvc::Subscribe 490
ISipGenericSvc::SendRequest 482	ISipMwiSvc::Unsubscribe 491
ISipGenericSvc::SetManager 482	ISipNotifierMgr 492
ISipGlareMgr 483	ISipNotifierMgr class 492
ISipGlareMgr class 483	EvExpired 492
EvReadyToRetry 483	EvFailure 493
ISipGlareMgr::EvReadyToRetry 483	EvFetched 493
ISipGlareSvc 484	EvInvalidSubscribe 494
ISipGlareSvc class 484	EvProgress 494
SetManager 485	EvRefreshed 495
	EvSubscribed 495

EvSuccess 496	ISipOptionTagsMgr::EvUnsupportedTag 505
EvTerminated 496	ISipOptionTagsSvc 96
ISipNotifierMgr::EvExpired 492	ISipOptionTagsSvc class 96
ISipNotifierMgr::EvFailure 493	AddSupportedExtension 97
ISipNotifierMgr::EvFetched 493	RemoveSupportedExtension 97
ISipNotifierMgr::EvInvalidSubscribe 494	SetCustomSupportedExtensions 98
ISipNotifierMgr::EvProgress 494	SetManager 98
ISipNotifierMgr::EvRefreshed 495	SetRequireHeaderVerification 99
ISipNotifierMgr::EvSubscribed 495	SetSupportedHeaderAddition 99
ISipNotifierMgr::EvSuccess 496	ISipOptionTagsSvc::AddSupportedExtension 97
ISipNotifierMgr::EvTerminated 496	ISipOptionTagsSvc::EBehavior 99
ISipNotifierSvc 497	ISipOptionTagsSvc::EBehavior enumeration 99
ISipNotifierSvc class 497	ISipOptionTagsSvc::RemoveSupportedExtension 97
AddEvent 499	ISipOptionTagsSvc::SetCustomSupportedExtensions 98
CreateSubscription 499	ISipOptionTagsSvc::SetManager 98
ExtendImplicitSubscription 500	ISipOptionTagsSvc::SetRequireHeaderVerification 99
GetCurrentSubscriptions 500	ISipOptionTagsSvc::SetSupportedHeaderAddition 99
Notify 501	ISipOutboundConnectionMgr 100
SetManager 502	ISipOutboundConnectionMgr class 100
SetMinimumExpiration 502	EvConnectionNeeded 100
Terminate 502	EvOutboundRequiredByPeer 101
ISipNotifierSvc::AddEvent 499	ISipOutboundConnectionMgr::EConnectionCreation 101
ISipNotifierSvc::CreateSubscription 499	ISipOutboundConnectionMgr::EConnectionCreation enumeration 101
ISipNotifierSvc::EReason 503	ISipOutboundConnectionMgr::EvConnectionNeeded 100
ISipNotifierSvc::EReason enumeration 503	ISipOutboundConnectionMgr::EvOutboundRequiredByPeer 101
ISipNotifierSvc::EState 504	ISipOutboundConnectionSvc 102
ISipNotifierSvc::EState enumeration 504	ISipOutboundConnectionSvc class 102
ISipNotifierSvc::EType 504	ConnectionCreationResult 103
ISipNotifierSvc::EType enumeration 504	GetPersistentConnectionsPreferredOrder 104
ISipNotifierSvc::ExtendImplicitSubscription 500	SetManager 104
ISipNotifierSvc::GetCurrentSubscriptions 500	SetPersistentConnectionsPreferredOrder 104
ISipNotifierSvc::Notify 501	ISipOutboundConnectionSvc::ConnectionCreationResult 103
ISipNotifierSvc::SetManager 502	ISipOutboundConnectionSvc::GetPersistentConnectionsPreferredOrder 104
ISipNotifierSvc::SetMinimumExpiration 502	ISipOutboundConnectionSvc::SetManager 104
ISipNotifierSvc::SSubscriptionInfo 498	ISipOutboundConnectionSvc::SetPersistentConnectionsPreferredOrder 104
ISipNotifierSvc::SSubscriptionInfo struct 498	ISipPersistentConnectionMgr 105
ISipNotifierSvc::Terminate 502	ISipPersistentConnectionMgr class 105
ISipOptionTagsMgr 504	
ISipOptionTagsMgr class 504	
EvUnsupportedTag 505	

EvConnectionEstablished 105	EvFailure 515
EvConnectionTerminated 106	EvProgress 515
EvErrorOnConnection 106	EvProtocolError 515
EvSendResult 106	EvSuccess 516
EvStartKeepAliveResult 107	ISipPublishMgr::EvConditionalRequestFailed 513
ISipPersistentConnectionMgr::EvConnectionEstablished 105	ISipPublishMgr::EvExpired 514
ISipPersistentConnectionMgr::EvConnectionTerminated 106	ISipPublishMgr::EvExpiresIntervalTooBrief 514
ISipPersistentConnectionMgr::EvErrorOnConnection 106	ISipPublishMgr::EvExpiring 514
ISipPersistentConnectionMgr::EvSendResult 106	ISipPublishMgr::EvFailure 515
ISipPersistentConnectionMgr::EvStartKeepAliveResult 107	ISipPublishMgr::EvProgress 515
ISipPersistentConnectionSvc 107	ISipPublishMgr::EvProtocolError 515
ISipPersistentConnectionSvc class 107	ISipPublishMgr::EvSuccess 516
ISipPrivacyMgr 505	ISipPublishSvc 516
ISipPrivacyMgr class 505	ISipPublishSvc class 516
EvPrivacyDnsResolutionCompleted 506	GetEntityTag 517
EvUncertifiedPrivacy 506	Modify 518
ISipPrivacyMgr::EvPrivacyDnsResolutionCompleted 506	Publish 518
ISipPrivacyMgr::EvUncertifiedPrivacy 506	Refresh 519
ISipPrivacySvc 507	Remove 520
ISipPrivacySvc class 507	SetDefaultExpiration 520
AddProxyRequire 509	SetEntityTag 521
EnableHeaderRemoval 509	SetExpiringThreshold 521
SetInstancePrivacyService 510	SetManager 522
SetManager 510	ISipPublishSvc::GetEntityTag 517
SetPrivacyType 511	ISipPublishSvc::Modify 518
SetSharedPrivacyService 511	ISipPublishSvc::Publish 518
ISipPrivacySvc::AddProxyRequire 509	ISipPublishSvc::Refresh 519
ISipPrivacySvc::EnableHeaderRemoval 509	ISipPublishSvc::Remove 520
ISipPrivacySvc::EPriValue 512	ISipPublishSvc::SetDefaultExpiration 520
ISipPrivacySvc::EPriValue enumeration 512	ISipPublishSvc::SetEntityTag 521
ISipPrivacySvc::SetInstancePrivacyService 510	ISipPublishSvc::SetExpiringThreshold 521
ISipPrivacySvc::SetManager 510	ISipPublishSvc::SetManager 522
ISipPrivacySvc::SetPrivacyType 511	ISipRedirectionMgr 522
ISipPrivacySvc::SetSharedPrivacyService 511	ISipRedirectionMgr class 522
ISipPublishMgr 513	~ISipRedirectionMgr 523
ISipPublishMgr class 513	EvRedirected 523
EvConditionalRequestFailed 513	ISipRedirectionMgr::~ISipRedirectionMgr 523
EvExpired 514	ISipRedirectionMgr::EvRedirected 523
EvExpiresIntervalTooBrief 514	ISipRedirectionSvc 523
EvExpiring 514	ISipRedirectionSvc class 523

GetContacts 525	ISipReferrerMgr 535
RemoveContact 525	ISipReferrerMgr class 535
SetManager 525	EvExpired 536
UseContact 526	EvExpiring 537
UseNextContact 526	EvIntervalTooSmall 537
ISipRedirectionSvc::GetContacts 525	EvInvalidNotify 537
ISipRedirectionSvc::RemoveContact 525	EvReferFailure 538
ISipRedirectionSvc::SetManager 525	EvReferProgress 539
ISipRedirectionSvc::UseContact 526	EvReferStatus 539
ISipRedirectionSvc::UseNextContact 526	EvReferSuccess 540
ISipRefereeMgr 527	EvSubscribeFailure 540
ISipRefereeMgr class 527	EvSubscribeProgress 541
EvExpired 528	EvSubscribeSuccess 541
EvFailure 528	ISipReferrerMgr::EState 542
EvInvalidRequest 528	ISipReferrerMgr::EState enumeration 542
EvProgress 529	ISipReferrerMgr::EvExpired 536
EvReferred 529	ISipReferrerMgr::EvExpiring 537
EvRefreshed 530	ISipReferrerMgr::EvIntervalTooSmall 537
EvSuccess 530	ISipReferrerMgr::EvInvalidNotify 537
EvTerminated 531	ISipReferrerMgr::EvReferFailure 538
ISipRefereeMgr::EvExpired 528	ISipReferrerMgr::EvReferProgress 539
ISipRefereeMgr::EvFailure 528	ISipReferrerMgr::EvReferStatus 539
ISipRefereeMgr::EvInvalidRequest 528	ISipReferrerMgr::EvReferSuccess 540
ISipRefereeMgr::EvProgress 529	ISipReferrerMgr::EvSubscribeFailure 540
ISipRefereeMgr::EvReferred 529	ISipReferrerMgr::EvSubscribeProgress 541
ISipRefereeMgr::EvRefreshed 530	ISipReferrerMgr::EvSubscribeSuccess 541
ISipRefereeMgr::EvSuccess 530	ISipReferrerSvc 542
ISipRefereeMgr::EvTerminated 531	ISipReferrerSvc class 542
ISipRefereeSvc 531	Refer 543
ISipRefereeSvc class 531	Refresh 543
ConfigureNotifyIdParameterUsage 532	SetExpiringThreshold 544
SendFinalReferralStatus 533	SetManager 545
SendReferralStatus 534	Terminate 545
SetManager 535	ISipReferrerSvc::Refer 543
ISipRefereeSvc::ConfigureNotifyIdParameterUsage 532	ISipReferrerSvc::Refresh 543
ISipRefereeSvc::EIdParameterUsage 535	ISipReferrerSvc::SetExpiringThreshold 544
ISipRefereeSvc::EIdParameterUsage enumeration 535	ISipReferrerSvc::SetManager 545
ISipRefereeSvc::SendFinalReferralStatus 533	ISipReferrerSvc::Terminate 545
ISipRefereeSvc::SendReferralStatus 534	ISipRegistrationMgr 546
ISipRefereeSvc::SetManager 535	ISipRegistrationMgr class 546

EvExpired 546	ISipRegistrationSvc::SetManager 556
EvExpiring 547	ISipRegistrationSvc::SetRegId 557
EvFailure 547	ISipRegistrationSvc::UpdateContact 557
EvProgress 548	ISipReliableProvisionalResponseMgr 559
EvSuccess 548	ISipReliableProvisionalResponseMgr class 559
ISipRegistrationMgr::EvExpired 546	EvInvalidPrack 559
ISipRegistrationMgr::EvExpiring 547	EvPrackFailure 560
ISipRegistrationMgr::EvFailure 547	EvPrackProgress 560
ISipRegistrationMgr::EvProgress 548	EvPrackSuccess 561
ISipRegistrationMgr::EvSuccess 548	EvReliableProvisionalResponseTimeout 561
ISipRegistrationSvc 548	EvReliableResponseReceived 561
ISipRegistrationSvc class 548	EvResponseAcknowledged 562
Add 550	ISipReliableProvisionalResponseMgr::EvInvalidPrack 559
AddLocalRegistration 551	ISipReliableProvisionalResponseMgr::EvPrackFailure 560
Clear 552	ISipReliableProvisionalResponseMgr::EvPrackProgress 560
GetContactMatchingType 553	ISipReliableProvisionalResponseMgr::EvPrackSuccess 561
GetList 553	ISipReliableProvisionalResponseMgr::EvReliableProvisionalResponseTimeout 561
GetRegId 554	ISipReliableProvisionalResponseMgr::EvReliableResponseReceived 561
Remove 554	ISipReliableProvisionalResponseMgr::EvResponseAcknowledged 562
SetContactMatchingType 555	ISipReliableProvisionalResponseSvc 562
SetExpiringThreshold 555	ISipReliableProvisionalResponseSvc class 562
SetIgnoreContactParamInSuccessResponse 556	IsReliabilityRequiredByPeer 563
SetManager 556	IsReliabilitySupportedByPeer 564
SetRegId 557	LocalRequestsRequireReliability 564
UpdateContact 557	MakeReliableServerEventControl 564
ISipRegistrationSvc::Add 550	Prack 565
ISipRegistrationSvc::AddLocalRegistration 551	SetManager 566
ISipRegistrationSvc::Clear 552	ISipReliableProvisionalResponseSvc::IsReliabilityRequiredByPeer 563
ISipRegistrationSvc::EContactMatchingType 558	ISipReliableProvisionalResponseSvc::IsReliabilitySupportedByPeer 564
ISipRegistrationSvc::EContactMatchingType enumeration 558	ISipReliableProvisionalResponseSvc::LocalRequestsRequireReliability 564
ISipRegistrationSvc::EThresholdType 558	ISipReliableProvisionalResponseSvc::MakeReliableServerEventControl 564
ISipRegistrationSvc::EThresholdType enumeration 558	ISipReliableProvisionalResponseSvc::Prack 565
ISipRegistrationSvc::GetContactMatchingType 553	ISipReliableProvisionalResponseSvc::SetManager 566
ISipRegistrationSvc::GetList 553	
ISipRegistrationSvc::GetRegId 554	
ISipRegistrationSvc::Remove 554	
ISipRegistrationSvc::SetContactMatchingType 555	
ISipRegistrationSvc::SetExpiringThreshold 555	
ISipRegistrationSvc::SetIgnoreContactParamInSuccessResponse 556	
	ISipReliableProvisionalResponseSvc::SetManager 566

ISipReplacesMgr 566	EvTerminationFailure 576
ISipReplacesMgr class 566	EvTerminationProgress 577
EvInvalidReplaces 567	EvTerminationSuccess 577
EvReplaces 567	
ISipReplacesMgr::EvInvalidReplaces 567	ISipSessionMgr::EvAcknowledged 570
ISipReplacesMgr::EvReplaces 567	ISipSessionMgr::EvFailure 571
ISipReplacesSvc 568	ISipSessionMgr::EvInvalidInvite 571
ISipReplacesSvc class 568	ISipSessionMgr::EvInviteCancelled 572
SetManager 569	ISipSessionMgr::EvInvited 573
ISipReplacesSvc::SetManager 569	ISipSessionMgr::EvInviteSuccessResponseTimeout 573
ISipServerEventControl 75	ISipSessionMgr::EvNewSessionNeededForOriginalInviteResponse 574
ISipServerEventControl class 75	ISipSessionMgr::EvProgress 574
GetOpaque 76	ISipSessionMgr::EvReInviteCancelled 574
SendResponse 76	ISipSessionMgr::EvReInvited 575
SetOpaque 77	ISipSessionMgr::EvSuccess 575
ISipServerEventControl::GetOpaque 76	ISipSessionMgr::EvTerminated 576
ISipServerEventControl::SendResponse 76	ISipSessionMgr::EvTerminationFailure 576
ISipServerEventControl::SetOpaque 77	ISipSessionMgr::EvTerminationProgress 577
ISipServerLocationSvc 108	ISipSessionMgr::EvTerminationSuccess 577
ISipServerLocationSvc class 108	ISipSessionStatefulProxyMgr 405
SetReqCtxServerLocationSvcMode 109	ISipSessionStatefulProxyMgr class 405
SetServerLocationListModifier 109	EvMaxForwardReached 406
ISipServerLocationSvc::EServerLocationSvcMode 109	EvNoTargetFound 406
ISipServerLocationSvc::EServerLocationSvcMode enumeration 109	EvRequestForwardTimeout 408
ISipServerLocationSvc::SetReqCtxServerLocationSvcMode 109	EvSessionCreated 409
ISipServerLocationSvc::SetServerLocationListModifier 109	EvSessionTerminated 409
ISipSessionMgr 569	EvTargetFound 410
ISipSessionMgr class 569	EvTransactionFailed 411
EvAcknowledged 570	EvTransactionProgress 412
EvFailure 571	EvTransactionSucceeded 413
EvInvalidInvite 571	ISipSessionStatefulProxyMgr::EvMaxForwardReached 406
EvInviteCancelled 572	ISipSessionStatefulProxyMgr::EvNoTargetFound 406
EvInvited 573	ISipSessionStatefulProxyMgr::EvRequestForwardTimeout 408
EvInviteSuccessResponseTimeout 573	ISipSessionStatefulProxyMgr::EvSessionCreated 409
EvNewSessionNeededForOriginalInviteResponse 574	ISipSessionStatefulProxyMgr::EvSessionTerminated 409
EvProgress 574	ISipSessionStatefulProxyMgr::EvTargetFound 410
EvReInviteCancelled 574	ISipSessionStatefulProxyMgr::EvTransactionFailed 411
EvReInvited 575	ISipSessionStatefulProxyMgr::EvTransactionProgress 412
EvSuccess 575	ISipSessionStatefulProxyMgr::EvTransactionSucceeded 413
EvTerminated 576	ISipSessionStatefulProxySvc 414

ISipSessionStatefulProxySvc class 414	ISipSessionTimerSvc class 584
AddLocalRecordRoute 416	GetMinSESec 585
AddRecordRoute 416	GetSessionExpiresSec 585
CancelAllClientTransactions 417	Reset 586
ForwardRequest 417	ResetSessionTimer 586
ForwardResponse 418	SetExpirationThresholds 586
SetConfig 419	SetManager 587
SetForkedDialogGrouperMgr 419	SetMinSESec 587
SetManager 420	SetMissingSessionExpiresInAnswerBehaviour 588
ISipSessionStatefulProxySvc::AddLocalRecordRoute 416	SetRefresher 588
ISipSessionStatefulProxySvc::AddRecordRoute 416	SetSessionExpiresSec 588
ISipSessionStatefulProxySvc::CancelAllClientTransactions 417	SetSessionTimerDemanded 589
ISipSessionStatefulProxySvc::ForwardRequest 417	ISipSessionTimerSvc::EMissingSessionExpireBehaviour 589
ISipSessionStatefulProxySvc::ForwardResponse 418	ISipSessionTimerSvc::EMissingSessionExpireBehaviour enumeration 589
ISipSessionStatefulProxySvc::SetConfig 419	ISipSessionTimerSvc::ERefresher 590
ISipSessionStatefulProxySvc::SetForkedDialogGrouperMgr 419	ISipSessionTimerSvc::ERefresher enumeration 590
ISipSessionStatefulProxySvc::SetManager 420	ISipSessionTimerSvc::GetMinSESec 585
ISipSessionSvc 578	ISipSessionTimerSvc::GetSessionExpiresSec 585
ISipSessionSvc class 578	ISipSessionTimerSvc::Reset 586
Ack 579	ISipSessionTimerSvc::ResetSessionTimer 586
Bye 580	ISipSessionTimerSvc::SetExpirationThresholds 586
HandleOriginalInviteResponseNewSession 580	ISipSessionTimerSvc::SetManager 587
Invite 581	ISipSessionTimerSvc::SetMinSESec 587
SetManager 582	ISipSessionTimerSvc::SetMissingSessionExpiresInAnswerBehaviour 588
ISipSessionSvc::Ack 579	ISipSessionTimerSvc::SetRefresher 588
ISipSessionSvc::Bye 580	ISipSessionTimerSvc::SetSessionExpiresSec 588
ISipSessionSvc::HandleOriginalInviteResponseNewSession 580	ISipSessionTimerSvc::SetSessionTimerDemanded 589
ISipSessionSvc::Invite 581	ISipStatelessDigestServerAuthSvc 110
ISipSessionSvc::SetManager 582	ISipStatelessDigestServerAuthSvc class 110
ISipSessionTimerMgr 582	ChallengeRequest 111
ISipSessionTimerMgr class 582	ContainsCredentials 112
EvSessionExpired 583	GetMd5AlgoHash 112
EvSessionIntervalTooShortRecv 583	QopMustBePresent 113
EvSessionIntervalTooShortSent 583	RejectRequest 113
EvSessionMustRefresh 584	SetAuthLifetime 114
ISipSessionTimerMgr::EvSessionExpired 583	SetPrivateKey 114
ISipSessionTimerMgr::EvSessionIntervalTooShortRecv 583	SetRequestedQualityOfProtection 114
ISipSessionTimerMgr::EvSessionIntervalTooShortSent 583	SetSupportedRealm 115
ISipSessionTimerMgr::EvSessionMustRefresh 584	
ISipSessionTimerSvc 584	

VerifyAuthentication 115	ISipStatelessProxySvc::SetManager 428
ISipStatelessDigestServerAuthSvc::ChallengeRequest 111	ISipStatisticsInfo 116
ISipStatelessDigestServerAuthSvc::ContainsCredentials 112	ISipStatisticsInfo class 116
ISipStatelessDigestServerAuthSvc::EAlgorithm 116	GetNumActiveTransactions 117
ISipStatelessDigestServerAuthSvc::EAlgorithm enumeration 116	GetNumDnsQueriesFailed 117
ISipStatelessDigestServerAuthSvc::EQualityOfProtection 116	GetNumDnsQueriesSucceeded 118
ISipStatelessDigestServerAuthSvc::EQualityOfProtection enumeration 116	GetNumFinalResponseRetransmissionsRx 118
ISipStatelessDigestServerAuthSvc::GetMd5AlgoHash 112	GetNumFinalResponseRetransmissionsTx 118
ISipStatelessDigestServerAuthSvc::QopMustBePresent 113	GetNumFinalResponseRx 118
ISipStatelessDigestServerAuthSvc::RejectRequest 113	GetNumFinalResponseTx 119
ISipStatelessDigestServerAuthSvc::SetAuthLifetime 114	GetNumNonFinalResponseRetransmissionsTx 119
ISipStatelessDigestServerAuthSvc::SetPrivateKey 114	GetNumNonFinalResponseRx 119
ISipStatelessDigestServerAuthSvc::SetRequestedQualityOfProtection 114	GetNumNonFinalResponseTx 120
ISipStatelessDigestServerAuthSvc::SetSupportedRealm 115	GetNumRequestRetransmissionsRx 120
ISipStatelessDigestServerAuthSvc::VerifyAuthentication 115	GetNumRequestRetransmissionsTx 120
ISipStatelessProxyMgr 420	GetNumRequestsRx 121
ISipStatelessProxyMgr class 420	GetNumRequestsTx 121
EvMaxForwardReached 421	GetTotalNumRequestsRx 121
EvNoTargetFound 421	GetTotalNumRequestsTx 121
EvResponseReceived 423	GetTotalNumResponsesRx 122
EvTargetFound 423	GetTotalNumResponsesTx 122
ISipStatelessProxyMgr::EvMaxForwardReached 421	Reset 122
ISipStatelessProxyMgr::EvNoTargetFound 421	ISipStatisticsInfo::GetNumActiveTransactions 117
ISipStatelessProxyMgr::EvResponseReceived 423	ISipStatisticsInfo::GetNumDnsQueriesFailed 117
ISipStatelessProxyMgr::EvTargetFound 423	ISipStatisticsInfo::GetNumDnsQueriesSucceeded 118
ISipStatelessProxySvc 425	ISipStatisticsInfo::GetNumFinalResponseRetransmissionsRx 118
ISipStatelessProxySvc class 425	ISipStatisticsInfo::GetNumFinalResponseRetransmissionsTx 118
AddLocalRecordRoute 426	ISipStatisticsInfo::GetNumFinalResponseRx 118
AddRecordRoute 426	ISipStatisticsInfo::GetNumFinalResponseTx 119
ForwardRequest 427	ISipStatisticsInfo::GetNumNonFinalResponseRetransmissionsTx 119
ForwardResponse 427	ISipStatisticsInfo::GetNumNonFinalResponseRx 119
SetConfig 428	ISipStatisticsInfo::GetNumNonFinalResponseTx 120
SetManager 428	ISipStatisticsInfo::GetNumRequestRetransmissionsRx 120
ISipStatelessProxySvc::AddLocalRecordRoute 426	ISipStatisticsInfo::GetNumRequestRetransmissionsTx 120
ISipStatelessProxySvc::AddRecordRoute 426	ISipStatisticsInfo::GetNumRequestsRx 121
ISipStatelessProxySvc::ForwardRequest 427	ISipStatisticsInfo::GetNumRequestsTx 121
ISipStatelessProxySvc::ForwardResponse 427	ISipStatisticsInfo::GetTotalNumRequestsRx 121
ISipStatelessProxySvc::SetConfig 428	ISipStatisticsInfo::GetTotalNumRequestsTx 121

ISipStatisticsInfo::GetTotalNumResponsesRx 122	SetManager 601
ISipStatisticsInfo::GetTotalNumResponsesTx 122	Subscribe 601
ISipStatisticsInfo::GetTotalNumTransactions 122	Terminate 602
ISipStatisticsInfo::Reset 122	ISipSubscriberSvc::AbortSubscription 596
ISipStatisticsSvc 123	ISipSubscriberSvc::AddEvent 597
ISipStatisticsSvc class 123	ISipSubscriberSvc::CreateSubscription 598
SetServiceMode 123	ISipSubscriberSvc::EType 603
SetTransactionStatistics 124	ISipSubscriberSvc::EType enumeration 603
ISipStatisticsSvc::EServiceMode 124	ISipSubscriberSvc::Fetch 599
ISipStatisticsSvc::EServiceMode enumeration 124	ISipSubscriberSvc::Refresh 599
ISipStatisticsSvc::SetServiceMode 123	ISipSubscriberSvc::SetExpiringThreshold 600
ISipStatisticsSvc::SetTransactionStatistics 124	ISipSubscriberSvc::SetManager 601
ISipSubscriberMgr 590	ISipSubscriberSvc::Subscribe 601
ISipSubscriberMgr class 590	ISipSubscriberSvc::Terminate 602
EvExpired 591	ISipSymmetricUdpSvc 124
EvExpiring 591	ISipSymmetricUdpSvc class 124
EvFailure 591	ISipTlsContextFactory 460
EvIntervalTooSmall 592	ISipTlsContextFactory class 460
EvInvalidNotify 592	AddTlsClientContextS 462
EvNotified 593	AddTlsServerContextS 462
EvProgress 594	GetDefaultTlsClientContextS 462
EvSuccess 594	GetDefaultTlsServerContextS 463
EvTerminated 595	GetTlsClientContextS 463
ISipSubscriberMgr::EvExpired 591	GetTlsServerContextS 464
ISipSubscriberMgr::EvExpiring 591	RemoveTlsClientContextS 464
ISipSubscriberMgr::EvFailure 591	RemoveTlsServerContextS 465
ISipSubscriberMgr::EvIntervalTooSmall 592	SetDefaultTlsClientContextS 465
ISipSubscriberMgr::EvInvalidNotify 592	SetDefaultTlsServerContextS 465
ISipSubscriberMgr::EvNotified 593	UpdateTlsClientContextS 466
ISipSubscriberMgr::EvProgress 594	UpdateTlsServerContextS 466
ISipSubscriberMgr::EvSuccess 594	ISipTlsContextFactory::AddTlsClientContextS 462
ISipSubscriberMgr::EvTerminated 595	ISipTlsContextFactory::AddTlsServerContextS 462
ISipSubscriberSvc 595	ISipTlsContextFactory::GetDefaultTlsClientContextS 462
ISipSubscriberSvc class 595	ISipTlsContextFactory::GetDefaultTlsServerContextS 463
AbortSubscription 596	ISipTlsContextFactory::GetTlsClientContextS 463
AddEvent 597	ISipTlsContextFactory::GetTlsServerContextS 464
CreateSubscription 598	ISipTlsContextFactory::RemoveTlsClientContextS 464
Fetch 599	ISipTlsContextFactory::RemoveTlsServerContextS 465
Refresh 599	ISipTlsContextFactory::SetDefaultTlsClientContextS 465
SetExpiringThreshold 600	ISipTlsContextFactory::SetDefaultTlsServerContextS 465

ISipTlsContextFactory::UpdateTlsClientContextS 466	ISipTransactionStatefulProxySvc::AddRecordRoute 437
ISipTlsContextFactory::UpdateTlsServerContextS 466	ISipTransactionStatefulProxySvc::CancelAllClientTransactions 438
ISipTransactionCompletionMgr 604	ISipTransactionStatefulProxySvc::ForwardRequest 438
ISipTransactionCompletionMgr class 604	ISipTransactionStatefulProxySvc::ForwardResponse 439
EvTransactionCompleted 604	ISipTransactionStatefulProxySvc::SetConfig 439
ISipTransactionCompletionMgr::EvTransactionCompleted 604	ISipTransactionStatefulProxySvc::SetManager 440
ISipTransactionCompletionSvc 605	ISipTransferMgr07 607
ISipTransactionCompletionSvc class 605	ISipTransferMgr07 class 607
CanEstablishNewTransaction 606	EvFinalReport 607
EnableParallelTransactions 606	EvFinalStatusRequired 608
SetManager 606	EvInvalidRequest 608
ISipTransactionCompletionSvc::CanEstablishNewTransaction 606	EvNotifyFailure 609
ISipTransactionCompletionSvc::EnableParallelTransactions 606	EvNotifyProgress 610
ISipTransactionCompletionSvc::SetManager 606	EvNotifySuccess 610
ISipTransactionStatefulProxyMgr 428	EvProgressReport 611
ISipTransactionStatefulProxyMgr class 428	EvReferFailure 611
EvMaxForwardReached 429	EvReferProgress 612
EvNoTargetFound 429	EvReferSuccess 612
EvRequestForwardTimeout 431	EvReportingExpired 613
EvTargetFound 431	EvTransferred 613
EvTransactionFailed 433	ISipTransferMgr07::EvFinalReport 607
EvTransactionProgress 434	ISipTransferMgr07::EvFinalStatusRequired 608
EvTransactionSucceeded 435	ISipTransferMgr07::EvInvalidRequest 608
ISipTransactionStatefulProxyMgr::EvMaxForwardReached 429	ISipTransferMgr07::EvNotifyFailure 609
ISipTransactionStatefulProxyMgr::EvNoTargetFound 429	ISipTransferMgr07::EvNotifyProgress 610
ISipTransactionStatefulProxyMgr::EvRequestForwardTimeout 431	ISipTransferMgr07::EvNotifySuccess 610
ISipTransactionStatefulProxyMgr::EvTargetFound 431	ISipTransferMgr07::EvProgressReport 611
ISipTransactionStatefulProxyMgr::EvTransactionFailed 433	ISipTransferMgr07::EvReferFailure 611
ISipTransactionStatefulProxyMgr::EvTransactionProgress 434	ISipTransferMgr07::EvReferProgress 612
ISipTransactionStatefulProxyMgr::EvTransactionSucceeded 435	ISipTransferMgr07::EvReferSuccess 612
ISipTransactionStatefulProxySvc 435	ISipTransferMgr07::EvReportingExpired 613
ISipTransactionStatefulProxySvc class 435	ISipTransferMgr07::EvTransferred 613
AddLocalRecordRoute 437	ISipTransferSvc07 614
AddRecordRoute 437	ISipTransferSvc07 class 614
CancelAllClientTransactions 438	ConfigureNotifyIdParameterUsage 615
ForwardRequest 438	ReportFailure 616
ForwardResponse 439	ReportFinalStatus 616
SetConfig 439	ReportPending 617
SetManager 440	ReportProgress 618
ISipTransactionStatefulProxySvc::AddLocalRecordRoute 437	ReportRefusal 619

SetManager	620	626
Transfer	621	ISipUaAssertedIdentityMgr::EvUntrustedProxy 627
ISipTransferSvc07::ConfigureNotifyIdParameterUsage	615	ISipUaAssertedIdentitySvc 628
ISipTransferSvc07::ReportFailure	616	ISipUaAssertedIdentitySvc class 628
ISipTransferSvc07::ReportFinalStatus	616	SetInstanceTrustedProxy 629, 630
ISipTransferSvc07::ReportPending	617	SetManager 631
ISipTransferSvc07::ReportProgress	618	SetPreferredIdentities 631
ISipTransferSvc07::ReportRefusal	619	SetSharedTrustedProxy 632, 633
ISipTransferSvc07::SetManager	620	ISipUaAssertedIdentitySvc::SetInstanceTrustedProxy 629, 630
ISipTransferSvc07::Transfer	621	ISipUaAssertedIdentitySvc::SetManager 631
ISipTransportObserver	467	ISipUaAssertedIdentitySvc::SetPreferredIdentities 631
ISipTransportObserver class	467	ISipUaAssertedIdentitySvc::SetSharedTrustedProxy 632, 633
EvConnectionClosed	468	ISipUpdateMgr 633
EvConnectionEstablished	468	ISipUpdateMgr class 633
EvObserverRemoved	468	EvFailure 634
EvPacketReceived	468	EvInvalidUpdate 635
EvTransportError	469	EvProgress 635
ISipTransportObserver::EClosureType	469	EvSuccess 636
ISipTransportObserver::EClosureType enumeration	469	EvUpdated 636
ISipTransportObserver::EInsertObserverPriority	469	ISipUpdateMgr::EvFailure 634
ISipTransportObserver::EInsertObserverPriority enumeration	469	ISipUpdateMgr::EvInvalidUpdate 635
ISipTransportObserver::EvConnectionClosed	468	ISipUpdateMgr::EvProgress 635
ISipTransportObserver::EvConnectionEstablished	468	ISipUpdateMgr::EvSuccess 636
ISipTransportObserver::EvObserverRemoved	468	ISipUpdateMgr::EvUpdated 636
ISipTransportObserver::EvPacketReceived	468	ISipUpdateSvc 637
ISipTransportObserver::EvTransportError	469	ISipUpdateSvc class 637
ISipTransportUser	470	SetManager 638
ISipTransportUser class	470	Update 638
EvCommandResult	470	ISipUpdateSvc::SetManager 638
ISipTransportUser::EvCommandResult	470	ISipUpdateSvc::Update 638
ISipUaAssertedIdentityMgr	623	ISipUserAgentSvc 639
ISipUaAssertedIdentityMgr class	623	ISipUserAgentSvc class 639
EvAssertedIdentity	624	AcceptOutOfOrderRemoteRequests 640
EvInvalidAssertedIdentity	625, 626	AddLocalContact 641
EvTrustedProxyDnsResolutionCompleted	626	CompleteDialogData 641
EvUntrustedProxy	627	Establish 642
ISipUaAssertedIdentityMgr::EvAssertedIdentity	624	EstablishForSentRequest 643
ISipUaAssertedIdentityMgr::EvInvalidAssertedIdentity	625, 626	GetCallId 643
ISipUaAssertedIdentityMgr::EvTrustedProxyDnsResolutionCompleted		GetCurrentTarget 644
		GetInstanceId 644

GetLocalAddr 644	ISipUserAgentSvc::GetCurrentTarget 644
GetLocalContact 645	ISipUserAgentSvc::GetInstanceId 644
GetLocalContactVector 645	ISipUserAgentSvc::GetLocalAddr 644
GetLocalDescriptorParameters 645	ISipUserAgentSvc::GetLocalContact 645
GetLocalSequenceNumber 646	ISipUserAgentSvc::GetLocalContactVector 645
GetLocalTag 646	ISipUserAgentSvc::GetLocalDescriptorParameters 645
GetPreloadedRoute 646	ISipUserAgentSvc::GetLocalSequenceNumber 646
GetRemoteAddr 647	ISipUserAgentSvc::GetLocalTag 646
GetRemoteDescriptorParameters 647	ISipUserAgentSvc::GetPreloadedRoute 646
GetRemoteSequenceNumber 648	ISipUserAgentSvc::GetRemoteAddr 647
GetRemoteSequenceNumber64bits 648	ISipUserAgentSvc::GetRemoteDescriptorParameters 647
GetRemoteTag 649	ISipUserAgentSvc::GetRemoteSequenceNumber 648
GetState 649	ISipUserAgentSvc::GetRemoteSequenceNumber64bits 648
SetCallId 649	ISipUserAgentSvc::GetRemoteTag 649
SetCurrentTarget 650	ISipUserAgentSvc::GetState 649
SetInstanceId 651	ISipUserAgentSvc::SetCallId 649
SetLocalAddr 651	ISipUserAgentSvc::SetCurrentTarget 650
SetLocalDescriptorParameters 652	ISipUserAgentSvc::SetInstanceId 651
SetLocalSequenceNumber 652	ISipUserAgentSvc::SetLocalAddr 651
SetLocalTag 653	ISipUserAgentSvc::SetLocalDescriptorParameters 652
SetPreloadedRoute 653	ISipUserAgentSvc::SetLocalSequenceNumber 652
SetRemoteAddr 654	ISipUserAgentSvc::SetLocalTag 653
SetRemoteDescriptorParameters 654	ISipUserAgentSvc::SetPreloadedRoute 653
SetRemoteSequenceNumber 655	ISipUserAgentSvc::SetRemoteAddr 654
SetRemoteSequenceNumber64bits 655	ISipUserAgentSvc::SetRemoteDescriptorParameters 654
SetRemoteTag 656	ISipUserAgentSvc::SetRemoteSequenceNumber 655
TerminateUsage 656	ISipUserAgentSvc::SetRemoteSequenceNumber64bits 655
UpdateRoute 657	ISipUserAgentSvc::SetRemoteTag 656
ISipUserAgentSvc::AcceptOutOfOrderRemoteRequests 640	ISipUserAgentSvc::TerminateUsage 656
ISipUserAgentSvc::AddLocalContact 641	ISipUserAgentSvc::UpdateRoute 657
ISipUserAgentSvc::CompleteDialogData 641	ISipViaManagementSvc 125
ISipUserAgentSvc::EPacketDirection 658	ISipViaManagementSvc class 125
ISipUserAgentSvc::EPacketDirection enumeration 658	SetViaManagementSvcMode 126
ISipUserAgentSvc::Establish 642	ISipViaManagementSvc::EViaManagementMode 126
ISipUserAgentSvc::EstablishForSentRequest 643	ISipViaManagementSvc::EViaManagementMode enumeration 126
ISipUserAgentSvc::EState 658	ISipViaManagementSvc::SetViaManagementSvcMode 126
ISipUserAgentSvc::EState enumeration 658	IUri 380
ISipUserAgentSvc::ETargetHeaderOption 658	IUri class 380
ISipUserAgentSvc::ETargetHeaderOption enumeration 658	~IUri 381
ISipUserAgentSvc::GetCallId 643	GenerateCopy 381

GetScheme	382	17
GetUriType	382	MXD_SIPSTACK_ENABLE_DEPRECATED_SIPTRANSACTION_INSTANCE_TRACKING
IsEquivalent	382	18
Parse	382	MXD_SIPSTACK_ENABLE_LOCALLY_GENERATED_ERROR_RESPONSE_DIFFERENTIATOR
Reset	383	15
Serialize	383	MXD_SIPSTACK_ENABLE_REGINFO_SUPPORT
IUri Constants	398	16
IUri::~IUri	381	MXD_SIPSTACK_ENABLE_SERVER_INVITE_SAME_BRANCH_ACK
IUri::ESpecialCharactersAllowed	383	14
IUri::ESpecialCharactersAllowed enumeration	383	MXD_SIPSTACK_ENABLE_SIP_CONNECTION_BLACKLIST_SVC_SUPPORT
IUri::EUriType	384	6
IUri::EUriType enumeration	384	MXD_SIPSTACK_ENABLE_SIP_CONNECTION_REUSE_SVC_SUPPORT
IUri::GenerateCopy	381	7
IUri::GetScheme	382	MXD_SIPSTACK_ENABLE_SIP_CORE_OUTPUT_CONTROLLING_SVC_SUPPORT
IUri::GetUriType	382	6
IUri::IsEquivalent	382	MXD_SIPSTACK_ENABLE_SIP_DESTINATION_SELECTION_SVC_SUPPORT
IUri::Parse	382	16
IUri::Reset	383	MXD_SIPSTACK_ENABLE_SIP_DIGEST_CLIENT_AUTH_SVC_SUPPORT
IUri::Serialize	383	9
M		MXD_SIPSTACK_ENABLE_SIP_DIVERSION_SVC_SUPPORT
M5T SIP SAFE	5	13
MxConvertSipHeader	389	MXD_SIPSTACK_ENABLE_SIP_ENUM_SVC_SUPPORT
MxConvertSipHeader function	389	15
MxConvertSipMethod	389, 390	MXD_SIPSTACK_ENABLE_SIP_GENERIC_SVC_SUPPORT
MxConvertSipMethod function	389, 390	9
MXD_POST_SIPSTACKCFG	5	MXD_SIPSTACK_ENABLE_SIP_GLARE_SVC_SUPPORT
MXD_SEQUENTIAL_TRANSACTION_TABLE	5	MXD_SIPSTACK_ENABLE_SIP_KEEP_ALIVE_SVC_SUPPORT
MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET		6
14		MXD_SIPSTACK_ENABLE_SIP_MWI_SVC_SUPPORT
MXD_SIPSTACK_DEFAULT_NUMBER_OF_HEADER_PER_PACKET		9
macro	14	MXD_SIPSTACK_ENABLE_SIP_NOTIFIER_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_DEFAULT_DATA_LOGGER	15	MXD_SIPSTACK_ENABLE_SIP_OPTION_TAGS_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPCONTEXT_INSTANCE_TRACKING		10
17		MXD_SIPSTACK_ENABLE_SIP_OUTBOUND_CONNECTION_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPPACKET_INSTANCE_TRACKING		7
17		MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_CURRENT_SRV_FQDN_RETRIEVAL
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		7
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		MXD_SIPSTACK_ENABLE_SIP_PERSISTENT_CONNECTION_SVC_SUPPORT
17		6
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		MXD_SIPSTACK_ENABLE_SIP_PRIVACY_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		10
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		MXD_SIPSTACK_ENABLE_SIP_PUBLISH_SVC_SUPPORT
17		10
MXD_SIPSTACK_ENABLE_DEPRECATED_SIPREQUEST_CONTEXT_INSTANCE_TRACKING		MXD_SIPSTACK_ENABLE_SIP_REDIRECTION_SVC_SUPPORT

MXD_SIPSTACK_ENABLE_SIP_REFEREE_SVC_SUPPORT	11	MXD_SIPSTACK_ENABLE_SIP_USER_AGENT_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_SIP_REFERRER_SVC_SUPPORT	11	13
MXD_SIPSTACK_ENABLE_SIP_REGISTRATION_SVC_SUPPORT	11	MXD_SIPSTACK_ENABLE_SIP_VIA_MANAGEMENT_SVC_SUPPORT
MXD_SIPSTACK_ENABLE_SIP_RELIABLE_PROVISIONAL_RESPONSE_SVC_SUPPORT	11	17
MXD_SIPSTACK_ENABLE_SIP_REPLACE_SVC_SUPPORT	11	MXD_SIPSTACK_ENABLE_SIPCONTEXT_INSTANCE_TRACKING
MXD_SIPSTACK_ENABLE_SIP_SERVER_LOCATION_SVC_SUPPORT	7	16
MXD_SIPSTACK_ENABLE_SIP_SERVER_MONITOR_SVC_SUPPORT	14	MXD_SIPSTACK_ENABLE_TLS
MXD_SIPSTACK_ENABLE_SIP_SESSION_STATEFUL_PROXY_SVC_SUPPORT	8	5
MXD_SIPSTACK_ENABLE_SIP_SESSION_SVC_SUPPORT	12	MXD_SIPSTACK_IPV6_ENABLE_SUPPORT
MXD_SIPSTACK_ENABLE_SIP_SESSION_TIMER_SVC_SUPPORT	12	14
MXD_SIPSTACK_ENABLE_SIP_SPIRALLING_SVC_SUPPORT	7	MXD_SIPSTACK_MAX_DUMP_LENGTH
MXD_SIPSTACK_ENABLE_SIP_STATELESS_DIGEST_SERVER_AUTH_SVC_SUPPORT	8	15
MXD_SIPSTACK_ENABLE_SIP_STATELESS_PROXY_SVC_SUPPORT	8	MXD_SIPSTACK_MAX_DUMP_LENGTH macro
MXD_SIPSTACK_ENABLE_SIP_STATISTICS_SVC_SUPPORT	15	MXD_SIPSTACK_SIP_TRANSACTION_PEER_ADDRESS_MATCH_BYPASS_ENABLE_SUPPORT
MXD_SIPSTACK_ENABLE_SIP_SUBSCRIBER_SVC_SUPPORT	12	16
MXD_SIPSTACK_ENABLE_SIP_SYMMETRIC_UDP_SVC_SUPPORT	8	MxGetDefaultReasonPhrase
MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_COMPLETION_SVC_SUPPORT	13	390
MXD_SIPSTACK_ENABLE_SIP_TRANSACTION_STATEFUL_PROXY_SVC_SUPPORT	8	MxGetDefaultReasonPhrase function
MXD_SIPSTACK_ENABLE_SIP_TRANSFER_SVC_07_SUPPORT	12	390
MXD_SIPSTACK_ENABLE_SIP_UA_ASSERTED_IDENTITY_SVC_SUPPORT	13	MxGetSipStatusClass
MXD_SIPSTACK_ENABLE_SIP_UA_HELPERS_SUPPORT	18	391
MXD_SIPSTACK_ENABLE_SIP_UPDATE_SVC_SUPPORT	13	MxGetSipStatusClass function
		391
		MxSetDefaultHeaderOrder
		391
		MxSetDefaultHeaderOrder function
		391
		mxt_PFNServerLocationListModifier
		126
		mxt_PFNServerLocationListModifier type
		126
		P
		pszHDRPARAM_BOUNDARY
		395
		pszHDRPARAM_COUNTER
		395
		pszHDRPARAM_LIMIT
		395
		pszHDRPARAM_LIMIT variable
		395
		pszHDRPARAM_LOCAL_TAG
		396
		pszHDRPARAM_LOCAL_TAG variable
		396
		pszHDRPARAM_OB
		396
		pszHDRPARAM_OB variable
		396
		pszHDRPARAM_PRIVACY
		395
		pszHDRPARAM_PRIVACY variable
		395
		pszHDRPARAM_REG_ID
		396
		pszHDRPARAM_REMOTE_TAG
		396
		pszHDRPARAM_SCREEN
		395
		pszHDRPARAM_SCREEN variable
		395
		pszHDRPARAM_SIP_INSTANCE
		396

pszHDRPARAM_SIP_INSTANCE variable 396
 pszNAMESPACE_GRUU 399
 pszREGINFO_ATTRIBUTE_AOR 399
 pszREGINFO_ATTRIBUTE_AOR variable 399
 pszREGINFO_ATTRIBUTE_CALLID 399
 pszREGINFO_ATTRIBUTE_CALLID variable 399
 pszREGINFO_ATTRIBUTE_CSEQ 399
 pszREGINFO_ATTRIBUTE_CSEQ variable 399
 pszREGINFO_ATTRIBUTE_DISPLAYNAME 399
 pszREGINFO_ATTRIBUTE_DISPLAYNAME variable 399
 pszREGINFO_ATTRIBUTE_DURATIONREGISTERED 399
 pszREGINFO_ATTRIBUTE_DURATIONREGISTERED variable 399
 pszREGINFO_ATTRIBUTE_EVENT 399
 pszREGINFO_ATTRIBUTE_EVENT variable 399
 pszREGINFO_ATTRIBUTE_EXPIRES 399
 pszREGINFO_ATTRIBUTE_EXPIRES variable 399
 pszREGINFO_ATTRIBUTE_ID 399
 pszREGINFO_ATTRIBUTE_ID variable 399
 pszREGINFO_ATTRIBUTE_NAME 399
 pszREGINFO_ATTRIBUTE_NAME variable 399
 pszREGINFO_ATTRIBUTE_RELATIVEPRIORITY 399
 pszREGINFO_ATTRIBUTE_RELATIVEPRIORITY variable 399
 pszREGINFO_ATTRIBUTE_RETRY_AFTER 399
 pszREGINFO_ATTRIBUTE_RETRY_AFTER variable 399
 pszREGINFO_ATTRIBUTE_STATE 399
 pszREGINFO_ATTRIBUTE_STATE variable 399
 pszREGINFO_ATTRIBUTE_VERSION 399
 pszREGINFO_ATTRIBUTE_XMLNS 399
 pszREGINFO_ATTRIBUTE_XMLNS variable 399
 pszREGINFO_ATTRIBUTE_XMLNSGR 399
 pszREGINFO_ATTRIBUTE_XMLNSGR variable 399
 pszREGINFO_CONTACT_EVENT_CREATED 400
 pszREGINFO_CONTACT_EVENT_CREATED variable 400
 pszREGINFO_CONTACT_EVENT_DEACTIVATED 400
 pszREGINFO_CONTACT_EVENT_DEACTIVATED variable 400
 pszREGINFO_CONTACT_EVENT_EXPIRED 400
 pszREGINFO_CONTACT_EVENT_EXPIRED variable 400
 pszREGINFO_CONTACT_EVENT_PROBATION 400
 pszREGINFO_CONTACT_EVENT_PROBATION variable 400
 pszREGINFO_CONTACT_EVENT_REFRESHED 400
 pszREGINFO_CONTACT_EVENT_REFRESHED variable 400
 pszREGINFO_CONTACT_EVENT_REGISTERED 400
 pszREGINFO_CONTACT_EVENT_REJECTED 400
 pszREGINFO_CONTACT_EVENT_REJECTED variable 400
 pszREGINFO_CONTACT_EVENT_SHORTENED 400
 pszREGINFO_CONTACT_EVENT_SHORTENED variable 400
 pszREGINFO_CONTACT_EVENT_UNREGISTERED 400
 pszREGINFO_CONTACT_EVENT_UNREGISTERED variable 400
 pszREGINFO_CONTACT_STATE_ACTIVE 400
 pszREGINFO_CONTACT_STATE_TERMINATED 400
 pszREGINFO_CONTACT_STATE_TERMINATED variable 400
 pszREGINFO_ELEMENT_CONTACT 400
 pszREGINFO_ELEMENT_CONTACT variable 400
 pszREGINFO_ELEMENT_PUBGRUU 400
 pszREGINFO_ELEMENT_PUBGRUU variable 400
 pszREGINFO_ELEMENT_REGINFO 400
 pszREGINFO_ELEMENT_REGISTRATION 400
 pszREGINFO_ELEMENT_REGISTRATION variable 400
 pszREGINFO_ELEMENT_TEMPGRUU 400
 pszREGINFO_ELEMENT_TEMPGRUU variable 400
 pszREGINFO_ELEMENT_UNKNONPARAM 400
 pszREGINFO_ELEMENT_UNKNONPARAM variable 400
 pszREGINFO_ELEMENT_URI 400
 pszREGINFO_ELEMENT_URI variable 400
 pszREGINFO_REGISTRATION_STATE_ACTIVE 400
 pszREGINFO_REGISTRATION_STATE_ACTIVE variable 400
 pszREGINFO_REGISTRATION_STATE_INIT 400
 pszREGINFO_REGISTRATION_STATE_TERMINATED 400
 pszREGINFO_REGISTRATION_STATE_TERMINATED variable 400
 pszREGINFO_STATE_FULL 400
 pszREGINFO_STATE_FULL variable 400
 pszREGINFO_STATE_PARTIAL 400
 pszX_M5T_ORIGIN_HEADER_NAME 401
 pszX_M5T_ORIGIN_HEADER_NAME variable 401
 pszX_M5T_ORIGIN_HEADER_VALUE 401
 pszX_M5T_ORIGIN_HEADER_VALUE variable 401

R

RegInfo Constants 399

Reginfo: Invalid values for contact's attributes 401
 Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo attribute names 399
 Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo element names 400
 Reginfo: RFC 3680 and draft-ietf-sipping-gruu-reg-event-08 reginfo namespaces 399
 Reginfo: RFC 3680 contact event values 400
 RFC 1847 parameter names 395
 RFC 3261 Header parameter names 396
 RFC 3261 Status codes 393
 RFC 3261 URI parameter names 395
 RFC 3265 (Event notification) status codes 392
 RFC 3265 Header parameter names 398
 RFC 3310 Header parameter names 397
 RFC 3326 Header parameter names 397
 RFC 3329 Header parameter names 396
 RFC 3455 Header parameter names 397
 RFC 3486 Header parameter names 398
 RFC 3486 URI parameter names 394
 RFC 3515 Subscription Events 659
 RFC 3581 Header parameter names 398
 RFC 3603 Header parameter names 397
 RFC 3680 contact state values 400
 RFC 3680 reginfo state values 400
 RFC 3680 registration state values 400
 RFC 3680 Subscription Events 659
 RFC 3842 Subscription Events 659
 RFC 3856 Subscription Events 659
 RFC 3857 Subscription Events 659
 RFC 3891 Header parameter names 398
 RFC 3910 Subscription Events 659
 RFC 3966 URI parameter names 395
 RFC 4538 Header parameter names 396
 RFC3261 "algorithm" header parameter values 397
 RFC3261 "qop" header parameter values 398
 RFC3261 "transport" URI parameter values 395
 RFC3261 "user" URI parameter values 395
 RFC3261 Magic cookie in "branch" parameter 397

S

Sip Event Constants 658
 SipCore 18
 SipCore tracing nodes 2
 SipCoreSvc 77
 SipCoreSvc tracing nodes 2
 SipParser 127
 SipParser tracing nodes 2
 SipProxy 401
 SipProxy tracing nodes 3
 SipTransaction tracing nodes 3
 SipTransport 440
 SipTransport tracing nodes 3
 SipUserAgent 471
 SipUserAgent tracing nodes 3
 Startup 660
 Status Codes 392
 szACCEPTABLE 392
 szACCEPTABLE variable 392
 szADDRESS_INCOMPLETE 393
 szADDRESS_INCOMPLETE variable 393
 szALTERNATIVE_SERVICE 393
 szALTERNATIVE_SERVICE variable 393
 szAMBIGUOUS 393
 szAMBIGUOUS variable 393
 szBAD_EVENT 392
 szBAD_EVENT variable 392
 szBAD_EXTENSION 393
 szBAD_EXTENSION variable 393
 szBAD_GATEWAY 393
 szBAD_GATEWAY variable 393
 szBAD_REQUEST 393
 szBAD_REQUEST variable 393
 szBAD_REQUEST_INVALID_CONTACT 393
 szBAD_REQUEST_INVALID_CONTACT variable 393
 szBAD_REQUEST_INVALID_MESSAGE_SUMMARY 393
 szBAD_REQUEST_INVALID_MESSAGE_SUMMARY variable 393
 szBUSY_EVERYWHERE 393

szBUSY_EVERYWHERE variable 393
szBUSY_HERE 393
szBUSY_HERE variable 393
szCALL_IS_BEING_FORWARDED 393
szCALL_IS_BEING_FORWARDED variable 393
szCALL_LEG_TRANSACTION_DOES_NOT_EXIST 393
szCALL_LEG_TRANSACTION_DOES_NOT_EXIST variable 393
szCONDITIONAL_REQUEST_FAILED 392
szCONDITIONAL_REQUEST_FAILED variable 392
szDECLINE 393
szDECLINE variable 393
szDOES_NOT_EXIST_ANYWHERE 393
szDOES_NOT_EXIST_ANYWHERE variable 393
szEXTENSION_REQUIRED 393
szEXTENSION_REQUIRED variable 393
szFORBIDDEN 393
szFORBIDDEN variable 393
szGLOBAL_NOT_ACCEPTABLE 393
szGLOBAL_NOT_ACCEPTABLE variable 393
szGONE 393
szGONE variable 393
szHDRPARAM_ALG 396
szHDRPARAM_ALGORITHM 396
szHDRPARAM_ALGORITHM_VALUE_MD5 397
szHDRPARAM_ALGORITHM_VALUE_MD5_SESS 397
szHDRPARAM_ALGORITHM_VALUE_MD5_SESS variable 397
szHDRPARAM_AUTS 397
szHDRPARAM_BRANCH 396
szHDRPARAM_BRANCH variable 396
szHDRPARAM_BRANCH_VALUE_MAGIC_COOKIE 397
szHDRPARAM_CALLED 397
szHDRPARAM_CALLING 397
szHDRPARAM_CALLING variable 397
szHDRPARAM_CAUSE 397
szHDRPARAM_CCF 397
szHDRPARAM_CGI_3GPP 397
szHDRPARAM_CGI_3GPP variable 397
szHDRPARAM_CHARGE 397
szHDRPARAM_CHARGE variable 397
szHDRPARAM_CNONCE 396
szHDRPARAM_CNONCE variable 396
szHDRPARAM_COMP 398
szHDRPARAM_CONTENT 397
szHDRPARAM_CONTENT variable 397
szHDRPARAM_COUNT 397
szHDRPARAM_COUNT variable 397
szHDRPARAM_D_ALG 396
szHDRPARAM_D_ALG variable 396
szHDRPARAM_D_QOP 396
szHDRPARAM_D_QOP variable 396
szHDRPARAM_D_VER 396
szHDRPARAM_D_VER variable 396
szHDRPARAM_DOMAIN 396
szHDRPARAM_DOMAIN variable 396
szHDRPARAM_DURATION 396
szHDRPARAM_DURATION variable 396
szHDRPARAM_EALG 396
szHDRPARAM_EALG variable 396
szHDRPARAM_EARLY_ONLY 398
szHDRPARAM_ECF 397
szHDRPARAM_ECF variable 397
szHDRPARAM_EXPIRES 396
szHDRPARAM_EXPIRES variable 396
szHDRPARAM_FROM_TAG 398
szHDRPARAM_FROM_TAG variable 398
szHDRPARAM_HANDLING 396
szHDRPARAM_HANDLING variable 396
szHDRPARAM_ICID_GENERATED_AT 397
szHDRPARAM_ICID_GENERATED_AT variable 397
szHDRPARAM_ICID_VALUE 397
szHDRPARAM_ICID_VALUE variable 397
szHDRPARAM_ID 398
szHDRPARAM_KEY 397
szHDRPARAM_KEY variable 397
szHDRPARAM_LOCROUTE 397
szHDRPARAM_LOCROUTE variable 397
szHDRPARAM_MADDR 396
szHDRPARAM_MADDR variable 396
szHDRPARAM_MOD 396
szHDRPARAM_MOD variable 396

szHDRPARAM_NC 396
szHDRPARAM_NC variable 396
szHDRPARAM_NEXTNONCE 396
szHDRPARAM_NEXTNONCE variable 396
szHDRPARAM_NONCE 396
szHDRPARAM_NONCE variable 396
szHDRPARAM_OPAQUE 396
szHDRPARAM_OPAQUE variable 396
szHDRPARAM_ORIG_IOI 397
szHDRPARAM_ORIG_IOI variable 397
szHDRPARAM_PORT1 396
szHDRPARAM_PORT1 variable 396
szHDRPARAM_PORT2 396
szHDRPARAM_PORT2 variable 396
szHDRPARAM_PROT 396
szHDRPARAM_PROT variable 396
szHDRPARAM_PURPOSE 396
szHDRPARAM_PURPOSE variable 396
szHDRPARAM_Q 396
szHDRPARAM_Q variable 396
szHDRPARAM_QOP 396
szHDRPARAM_QOP variable 396
szHDRPARAM_QOP_VALUE_AUTH 398
szHDRPARAM_QOP_VALUE_AUTH_INT 398
szHDRPARAM_QOP_VALUE_AUTH_INT variable 398
szHDRPARAM_REALM 396
szHDRPARAM_REALM variable 396
szHDRPARAM_REASON 398
szHDRPARAM_REASON variable 398
szHDRPARAM_RECEIVED 396
szHDRPARAM_RECEIVED variable 396
szHDRPARAM_REDIRECTOR_URI 397
szHDRPARAM_REDIRECTOR_URI variable 397
szHDRPARAM_REFRESHER 398
szHDRPARAM_REFRESHER_VALUE_UAC 398
szHDRPARAM_REFRESHER_VALUE_UAC variable 398
szHDRPARAM_REFRESHER_VALUE_UAS 398
szHDRPARAM_REFRESHER_VALUE_UAS variable 398
szHDRPARAM_RESPONSE 396
szHDRPARAM_RESPONSE variable 396
szHDRPARAM_RETRY_AFTER 398
szHDRPARAM_RETRY_AFTER variable 398
szHDRPARAM_RKSGROUP 397
szHDRPARAM_RKSGROUP variable 397
szHDRPARAM_ROUTING 397
szHDRPARAM_ROUTING variable 397
szHDRPARAM_RPORT 398
szHDRPARAM_RSPAUTH 396
szHDRPARAM_RSPAUTH variable 396
szHDRPARAM_SPI 396
szHDRPARAM_SPI variable 396
szHDRPARAM_STALE 396
szHDRPARAM_STALE variable 396
szHDRPARAM_TAG 396
szHDRPARAM_TAG variable 396
szHDRPARAM_TERM_IOI 397
szHDRPARAM_TERM_IOI variable 397
szHDRPARAM_TEXT 397
szHDRPARAM_TEXT variable 397
szHDRPARAM_TO_TAG 398
szHDRPARAM_TO_TAG variable 398
szHDRPARAM_TTL 396
szHDRPARAM_TTL variable 396
szHDRPARAM_URI 396
szHDRPARAM_URI variable 396
szHDRPARAM_USERNAME 396
szHDRPARAM_USERNAME variable 396
szHDRPARAM_UTRAN_CELL_ID_3GPP 397
szHDRPARAM_UTRAN_CELL_ID_3GPP variable 397
szINTERNAL_SERVER_ERROR 393
szINTERNAL_SERVER_ERROR variable 393
szINTERVAL_TOO_BRIEF 393
szINTERVAL_TOO_BRIEF variable 393
szINVALID_P_ASSERTED_IDENTITY 393
szINVALID_P_ASSERTED_IDENTITY variable 393
szINVALID_SUBSCRIPTION_STATE 393
szINVALID_SUBSCRIPTION_STATE variable 393
szLOOP_DETECTED 393
szLOOP_DETECTED variable 393
szMESSAGE_TOO_LARGE 393

szMESSAGE_TOO_LARGE variable 393
szMETHOD_NOT_ALLOWED 393
szMETHOD_NOT_ALLOWED variable 393
szMISSING_SUBSCRIPTION_STATE 393
szMISSING_SUBSCRIPTION_STATE variable 393
szMOVED_PERMANENTLY 393
szMOVED_PERMANENTLY variable 393
szMOVED_TEMPORARILY 393
szMOVED_TEMPORARILY variable 393
szMULTIPLE_CHOICES 393
szMULTIPLE_CHOICES variable 393
szNOT_ACCEPTABLE 393
szNOT_ACCEPTABLE variable 393
szNOT_ACCEPTABLE_HERE 393
szNOT_ACCEPTABLE_HERE variable 393
szNOT_FOUND 393
szNOT_FOUND variable 393
szNOT_IMPLEMENTED 393
szNOT_IMPLEMENTED variable 393
szOK 393
szOK variable 393
szPAYMENT_REQUIRED 393
szPAYMENT_REQUIRED variable 393
szPROVIDE_REFERRER_IDENTITY 392
szPROVIDE_REFERRER_IDENTITY variable 392
szPROXY_AUTHENTICATION_REQUIRED 393
szPROXY_AUTHENTICATION_REQUIRED variable 393
szQUEUED 393
szQUEUED variable 393
szREQUEST_ENTITY_TOO_LARGE 393
szREQUEST_ENTITY_TOO_LARGE variable 393
szREQUEST_PENDING 393
szREQUEST_PENDING variable 393
szREQUEST_TERMINATED 393
szREQUEST_TERMINATED variable 393
szREQUEST_TIMEOUT 393
szREQUEST_TIMEOUT variable 393
szREQUEST_URI_TOO_LARGE 393
szREQUEST_URI_TOO_LARGE variable 393
szRINGING 393
szRINGING variable 393
szSERVER_TIME_OUT 393
szSERVER_TIME_OUT variable 393
szSERVICE_UNAVAILABLE 393
szSERVICE_UNAVAILABLE variable 393
szSESSION_INTERVAL_TOO_SMALL 393
szSESSION_INTERVAL_TOO_SMALL variable 393
szSESSION_PROGRESS 393
szSESSION_PROGRESS variable 393
szSIP_EVENT_MESSAGE_SUMMARY 659
szSIP_EVENT_PRESENCE 659
szSIP_EVENT_REFER 659
szSIP_EVENT_REG 659
szSIP_EVENT_SPIRITS_INDPS 659
szSIP_EVENT_SPIRITS_USER_PROF 659
szSIP_EVENT_SPIRITS_USER_PROF variable 659
szSIP_EVENT_WINFO 659
szSIP_VERSION_NOT_SUPPORTED 393
szSIP_VERSION_NOT_SUPPORTED variable 393
szTEMPORARILY_NOT_AVAILABLE 393
szTEMPORARILY_NOT_AVAILABLE variable 393
szTOO_MANY_HOPS 393
szTOO_MANY_HOPS variable 393
szTRYING 393
szTRYING variable 393
szUNAUTHORIZED 393
szUNAUTHORIZED variable 393
szUNDECIPHERABLE 393
szUNDECIPHERABLE variable 393
szUNKNOWN_RESOURCE_PRIORITY 394
szUNKNOWN_RESOURCE_PRIORITY variable 394
szUNSUPPORTED_MEDIA_TYPE 393
szUNSUPPORTED_MEDIA_TYPE variable 393
szUNSUPPORTED_REFERER_TO_URI_SCHEME 393
szUNSUPPORTED_REFERER_TO_URI_SCHEME variable 393
szUNSUPPORTED_URI_SCHEME 393
szUNSUPPORTED_URI_SCHEME variable 393
szURIPARAM_COMP 394
szURIPARAM_COMP_VALUE_SIGCOMP 394
szURIPARAM_COMP_VALUE_SIGCOMP variable 394

szURIPARAM_LR	395	uADDRESS_INCOMPLETE	393
szURIPARAM_MADDR	395	uADDRESS_INCOMPLETE variable	393
szURIPARAM_MADDR	variable 395	uALTERNATIVE_SERVICE	393
szURIPARAM_METHOD	395	uALTERNATIVE_SERVICE variable	393
szURIPARAM_METHOD	variable 395	uAMBIGUOUS	393
szURIPARAM_PHONE_CONTEXT	395	uAMBIGUOUS variable	393
szURIPARAM_TRANSPORT	395	uBAD_EVENT	392
szURIPARAM_TRANSPORT	variable 395	uBAD_EVENT variable	392
szURIPARAM_TRANSPORT_TCP	395	uBAD_EXTENSION	393
szURIPARAM_TRANSPORT_TCP	variable 395	uBAD_EXTENSION variable	393
szURIPARAM_TRANSPORT_TLS	395	uBAD_GATEWAY	393
szURIPARAM_TRANSPORT_TLS	variable 395	uBAD_GATEWAY variable	393
szURIPARAM_TRANSPORT_UDP	395	uBAD_REQUEST	393
szURIPARAM_TTL	395	uBAD_REQUEST variable	393
szURIPARAM_TTL	variable 395	uBUSY_EVERYWHERE	393
szURIPARAM_USER	395	uBUSY_EVERYWHERE variable	393
szURIPARAM_USER	variable 395	uBUSY_HERE	393
szURIPARAM_USER_VALUE_IP	395	uBUSY_HERE variable	393
szURIPARAM_USER_VALUE_PHONE	395	uCALL_IS_BEING_FORWARDED	393
szURIPARAM_USER_VALUE_PHONE	variable 395	uCALL_IS_BEING_FORWARDED variable	393
szURISCHEME_IM	399	uCALL_LEG_TRANSACTION_DOES_NOT_EXIST	393
szURISCHEME_IM	variable 399	uCALL_LEG_TRANSACTION_DOES_NOT_EXIST variable	393
szURISCHEME_PRES	399	uCONDITIONAL_REQUEST_FAILED	392
szURISCHEME_PRES	variable 399	uDECLINE	393
szURISCHEME_SIP	399	uDECLINE variable	393
szURISCHEME_SIP	variable 399	uDOES_NOT_EXIST_ANYWHERE	393
szURISCHEME_SIPS	399	uDOES_NOT_EXIST_ANYWHERE variable	393
szURISCHEME_SIPS	variable 399	uEXTENSION_REQUIRED	393
szURISCHEME_TEL	399	uEXTENSION_REQUIRED variable	393
szURISCHEME_TEL	variable 399	uFORBIDDEN	393
szUSE_PROXY	393	uFORBIDDEN variable	393
szUSE_PROXY	variable 393	uGLOBAL_NOT_ACCEPTABLE	393
		uGLOBAL_NOT_ACCEPTABLE variable	393
		uGONE	393
		uGONE variable	393
		uHEADERS_PARSE_ALL	401
		uHEADERS_PARSE_ALL variable	401
		uINTERNAL_SERVER_ERROR	393
		uINTERNAL_SERVER_ERROR variable	393
		uINTERVAL_TOO_BRIEF	393

T

Tracing Nodes 2

Types 126

U

uACCEPTABLE 392

uINTERVAL_TOO_BRIEF variable 393
uLOOP_DETECTED 393
uLOOP_DETECTED variable 393
uMESSAGE_TOO_LARGE 393
uMESSAGE_TOO_LARGE variable 393
uMETHOD_NOT_ALLOWED 393
uMETHOD_NOT_ALLOWED variable 393
uMOVED_PERMANENTLY 393
uMOVED_PERMANENTLY variable 393
uMOVED_TEMPORARILY 393
uMOVED_TEMPORARILY variable 393
uMULTIPLE_CHOICES 393
uMULTIPLE_CHOICES variable 393
uNOT_ACCEPTABLE 393
uNOT_ACCEPTABLE variable 393
uNOT_ACCEPTABLE_HERE 393
uNOT_ACCEPTABLE_HERE variable 393
uNOT_FOUND 393
uNOT_FOUND variable 393
uNOT_IMPLEMENTED 393
uNOT_IMPLEMENTED variable 393
uOK 393
uOK variable 393
uPAYMENT_REQUIRED 393
uPAYMENT_REQUIRED variable 393
uPROVIDE_REFERRER_IDENTITY 392
uPROXY_AUTHENTICATION_REQUIRED 393
uPROXY_AUTHENTICATION_REQUIRED variable 393
uQUEUED 393
uQUEUED variable 393
uREGINFO_CONTACT_INVALID_CSEQ 401
uREGINFO_CONTACT_INVALID_CSEQ variable 401
uREGINFO_CONTACT_INVALID_DURATION_REGISTERED 401
uREGINFO_CONTACT_INVALID_EXPIRES 401
uREGINFO_CONTACT_INVALID_EXPIRES variable 401
uREGINFO_CONTACT_INVALID_RETRY_AFTER 401
uREGINFO_CONTACT_INVALID_RETRY_AFTER variable 401
uREQUEST_ENTITY_TOO_LARGE 393
uREQUEST_ENTITY_TOO_LARGE variable 393
uREQUEST_PENDING 393
uREQUEST_PENDING variable 393
uREQUEST_TERMINATED 393
uREQUEST_TERMINATED variable 393
uREQUEST_TIMEOUT 393
uREQUEST_TIMEOUT variable 393
uREQUEST_URI_TOO_LARGE 393
uREQUEST_URI_TOO_LARGE variable 393
Uri Parameter 394
uRINGING 393
uRINGING variable 393
uSERVER_TIME_OUT 393
uSERVER_TIME_OUT variable 393
uSERVICE_UNAVAILABLE 393
uSERVICE_UNAVAILABLE variable 393
uSESSION_INTERVAL_TOO_SMALL 393
uSESSION_PROGRESS 393
uSESSION_PROGRESS variable 393
uSIP_EVENT_MESSAGE_SUMMARY_DEFAULT_S 659
uSIP_EVENT_MESSAGE_SUMMARY_DEFAULT_S variable 659
uSIP_EVENT_PRESENCE_DEFAULT_S 659
uSIP_EVENT_PRESENCE_DEFAULT_S variable 659
uSIP_EVENT_REFER_DEFAULT_S 659
uSIP_EVENT_REFER_DEFAULT_S variable 659
uSIP_EVENT_REG_DEFAULT_S 659
uSIP_EVENT_REG_DEFAULT_S variable 659
uSIP_EVENT_SPIRITS_INDPS_DEFAULT_S 659
uSIP_EVENT_SPIRITS_INDPS_DEFAULT_S variable 659
uSIP_EVENT_SPIRITS_USER_PROF_DEFAULT_S 659
uSIP_EVENT_SPIRITS_USER_PROF_DEFAULT_S variable 659
uSIP_EVENT_WINFO_DEFAULT_S 659
uSIP_EVENT_WINFO_DEFAULT_S variable 659
uSIP_VERSION_NOT_SUPPORTED 393
uSIP_VERSION_NOT_SUPPORTED variable 393
uTEMPORARILY_NOT_AVAILABLE 393
uTEMPORARILY_NOT_AVAILABLE variable 393
uTOO_MANY_HOPS 393
uTOO_MANY_HOPS variable 393
uTRYING 393
uUNAUTHORIZED 393
uUNAUTHORIZED variable 393

uUNDECIPHERABLE 393
uUNDECIPHERABLE variable 393
uUNKNOWN_RESOURCE_PRIORITY 394
uUNSUPPORTED_MEDIA_TYPE 393
uUNSUPPORTED_MEDIA_TYPE variable 393
uUNSUPPORTED_URI_SCHEME 393
uUNSUPPORTED_URI_SCHEME variable 393
uUSE_PROXY 393
uUSE_PROXY variable 393

V

Variables 391, 658