



Dialogic® NaturalAccess™ SIP for NaturalCallControl™ API Developer's Manual

Copyright and legal notices

Copyright © 2005-2010 Dialogic Corporation. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
9000-62449-10	February 2005	SRR, Release 1.0 Beta
9000-62449-11	March 2005	SRR, Release 1.0 Beta 2
9000-62449-12	May 2005	SRG, Release 1.0 GA
9000-62449-13	August 2006	DEH, Release 1.1
9000-62449-14	February 2009	LBG, Release 2.0
64-0516-01	October 2009	LBG, NaturalAccess R9.0
64-0516-02	December 2009	LBG, NaturalAccess R9.0.1
64-0516-03	September 2010	LBG, NaturalAccess R9.0.4
Last modified: 2010-08-19		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1. Introduction.....	7
2. Session Initiation Protocol (SIP)	10
SIP overview	10
Protocol layers	10
NaturalAccess SIP for NCC API software components	11
readme file	11
NaturalAccess SIP server with SIP stack	11
NCC libraries for SIP	11
Header files	11
Demonstration program	12
Configuration files.....	12
3. Developing NaturalAccess SIP applications	13
Running SIP for NCC API	13
Setting up the environment	13
NaturalAccess features	14
Contexts and event queues.....	14
NaturalAccess services	14
Fusion	14
Building a SIP application	15
Implementing media streaming	16
4. Starting an SIP server.....	17
Configuring the SIP server.....	17
Command line options.....	17
Configuration parameters	19
Sample configuration file	26
Running a SIP server	26
Logging.....	27
5. Establishing communication sessions	28
Exchanging SIP requests and responses	28
Requests	28
Responses	28
Establishing a session	29
Terminating a session	29
Sending SIP messages.....	29
Using SIP for NCC API functions that send SIP messages automatically	30
Using nccCallMessage	31
Receiving SIP messages	31
Including information in the buffer attached to an associated NCC event	32
Including information in the buffer attached to NCCEVN_PROTOCOL_EVENT	33
Receiving SIP events	33
6. SIP data and messages	34
SIP for NCC API functions.....	34
Data structure formats	34
Creating data structures	35
SIP message formats	35
SIP information elements	36
SIP information element values.....	40
SIP_IE_SDP_MEDIA_TYPE	40

SIP_IE_SDP_MEDIA_PORT	40
SIP_IE_SDP_MEDIA_PROTOCOL	41
SIP_IE_SDP_ENCODING	41
SIP error codes	41
7. Using the NCC API	44
NCC API functions and SIP for NCC API	44
Automatic responses	47
Placing a call	47
Prototype	48
SIP message information structure	48
Examples	50
Answering a call	52
Accepting a call	53
Rejecting a call	54
Disconnecting a call	55
Getting call status information	55
Getting extended call status information	56
Acknowledging a call	57
Transferring a call	58
Placing on hold and retrieving a call	59
Sending a PRACK	61
Acknowledging a PRACK	61
Accepting or rejecting an incoming transfer request	62
Registering a user	62
Prototype	62
SIP registration information structure	63
Example	63
Requesting capability information	63
Responding with capability information	64
SIP capability information structure	65
NCC API events	66
CTA_EVENT structure	66
Sequence diagrams	69
Inbound calls	70
Outbound calls	71
Local disconnect	72
Remote disconnect	73
Capability mask	73
8. Using the MCC SDP library	75
Overview of the MCC SDP library	75
Reading an SDP message	75
Creating an SDP message	76
Adding session-level information to the SDP message	76
Adding media descriptions to the SDP message	77
Creating an SDP message example	78
Printing an SDP message	79
9. MCC SDP functions	80
Using the MCC SDP function reference	80
mccdPrintSDP	80
mccdReadSDP	82
mccdSdpAddAttribute	83
mccdSdpAddConnIP4	84

mccdSdpAddConnIP6	84
mccdSdpAddInformation	85
mccdSdpAddMediaBegin	86
mccdSdpAddMediaEnd	87
mccdSdpAddMediaFormat	87
mccdSdpAddRtpmap	88
mccdSdpAddSessionIP4	89
mccdSdpAddSessionIP6	89
mccdSdpGetLength	90
mccdSdpInit	91
10. MCC SDP structures	92
Using the MCC SDP structure reference	92
MCC_SDP	93
MCC_SDP_ADDRESS	95
MCC_SDP_ATTRIBUTE	95
MCC_SDP_FORMAT	96
MCC_SDP_MEDIA	97
MCC_SDP_RTPMAP	99
11. Using the demonstration program	101
Interactive test: ctasip	101
ctasip commands	102
ctasip parameters	105
Outbound call example	108
Inbound call example	109
12. Index	111

1. Introduction

The *Dialogic® NaturalAccess™ SIP for NaturalCallControl™ API Developer's Manual* explains how to develop a Session Initiation Protocol (SIP) application using NaturalAccess for NCC SIP software. It provides information about building SIP applications that perform call control with the Natural Access Call Control (NCC) API.

This document is for telephony and VoIP application developers who use Natural Access. It defines telephony and IP terms where applicable, but assumes that you are familiar with these concepts and the C programming language.

Terminology

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

Former terminology	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System

Former terminology	Dialogic terminology
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API

Former terminology	Dialogic terminology
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel
Video Access Utilities	Dialogic® NaturalAccess™ Video Access Toolkit Utilities
Video Mail Application Demonstration Program	Dialogic® NaturalAccess™ Video Access Toolkit Video Mail Application Demonstration Program
Video Messaging Server Interface	Dialogic® NaturalAccess™ Video Access Toolkit Video Messaging Server Interface
3G-324M Interface	Dialogic® NaturalAccess™ Video Access Toolkit 3G-324M Interface

2. Session Initiation Protocol (SIP)

SIP overview

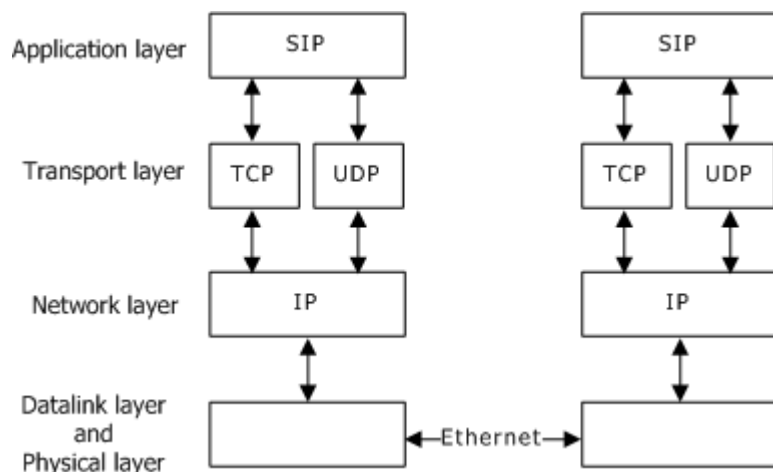
The Session Initiation Protocol (SIP) is an internet protocol that enables two or more users to establish a communication path and exchange data. This exchange of data is called a session. The data can be audio, video, or text. SIP enables users to identify each other, establish a connection, agree on a data format, exchange information, and end the communication.

Protocol layers

SIP is a layered protocol as defined in RFC 3261. Each layer provides a processing stage with defined functionality. The following table describes these layers:

Layer	Description
Application layer	Provides an interface to an application that enables the application to communicate with other applications.
Transport layer	Splits data into packets for transmission and reassembles them when the packets are received. Hides the details of transport from the upper layers.
Network layer	Packages the data with the correct address information.
Datalink layer and physical layer	The datalink layer creates an error free transmission channel. The physical layer moves the actual data from one point to another.

The following illustration shows the SIP layers:



The functionality of each layer includes the services and functions of all of the layers below it.

NaturalAccess SIP for NCC API software components

NaturalAccess SIP for NCC API contains the following components:

- [readme file](#)
- [NaturalAccess SIP server with SIP stack](#)
- [NCC libraries for SIP](#)
- [Header files](#)
- [Demonstration program](#)
- Sample [configuration files](#)

readme file

The *readme file* is an ASCII text file that contains release information that does not appear in other documentation. The file is named *readme_nccsip.txt*.

NaturalAccess SIP server with SIP stack

The following files are provided with the SIP for NCC API implementation:

Operating system	File name
Windows	<i>nmssip.exe</i>
UNIX	<i>nmssip</i>

NCC libraries for SIP

The NCC libraries run on the host. The application uses them to interact with the SIP for NCC API stack and to communicate with the NCC API.

Operating system	File name
Windows	<i>sipmgr.dll</i>
UNIX	<i>libsipmgr.so</i>

Header files

The following header file is supplied with SIP for NCC API software:

File name	Description
<i>nccxsip.h</i>	SIP specific header file

Demonstration program

ctasip is a demonstration program that accepts keyboard commands and enables users to perform the following tasks:

- Register SIP users
- Place SIP calls
- Answer incoming SIP calls
- Terminate SIP calls

This demonstration program supports CG Series boards.

Operating system	File name
Windows	<i>ctasip.exe</i>
UNIX	<i>ctasip</i>

Configuration files

The following configuration files are included with the SIP for NCC API software:

- *ctasip.cfg*
- *ctasipHmp.cfg*
- *ctasipFusion.cfg*
- *ctasipFusion2.cfg*
- *cg6kfusion-sip.cfg*

These files are located in the *ctasip* directory except for the *cg6kfusion-sip.cfg* file. This file is located in the *nms\cg\load* directory (Windows) and in the *nms/cg/load* directory (UNIX).

3. Developing NaturalAccess SIP applications

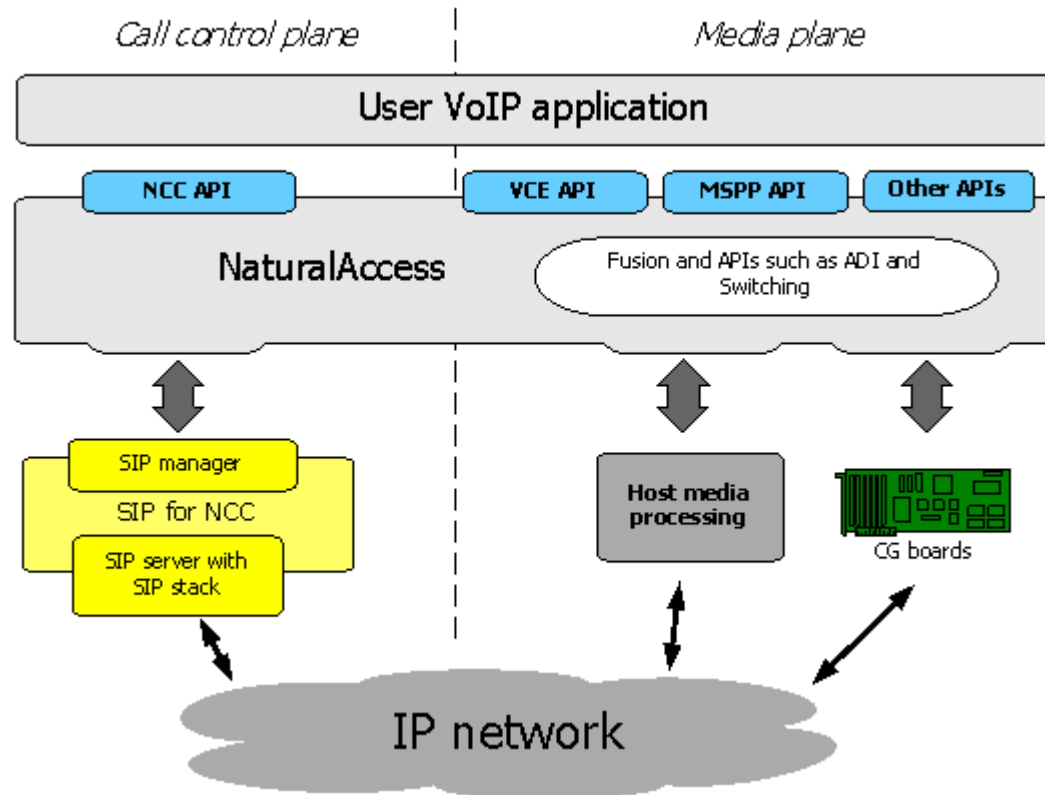
Running SIP for NCC API

The NaturalAccess SIP for NCC API runs on the host computer and not on NaturalAccess boards. Because of this, it can operate seamlessly with Fusion running on CG boards.

Setting up the environment

Before building applications using the SIP for NCC API, install NaturalAccess and SIP for NCC with a valid license on your system. SIP for NCC applications use the NaturalAccess development environment for exchanging data with a SIP for NCC server.

The following illustration shows an overview of NaturalAccess components and how they relate to the SIP implementation:



NaturalAccess features

NaturalAccess provides:

- A standard hardware-independent application programming interface.
- Sets of call control and media functions grouped into logical services.
- An interface for accessing and changing service parameters.
- Error trapping features.
- An architecture that supports as-needed resource allocation.
- Support for both single-threaded and multi-threaded programming models to meet a variety of application program requirements.
- Multiple operating system support.

Contexts and event queues

NaturalAccess organizes services and accompanying resources around a single processing context. A context usually represents an application instance controlling a single call.

An event queue is the communication path from a NaturalAccess service to an application. A NaturalAccess service generates events that indicate specific conditions or state changes and sends them to applications through the event queue.

NaturalAccess services

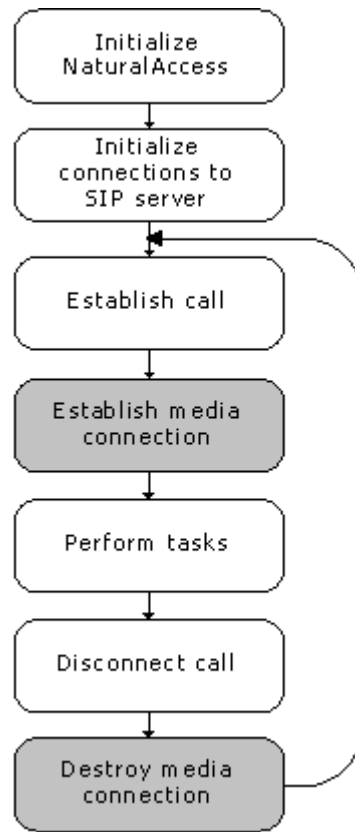
NaturalAccess services provide functions for establishing and maintaining network connections, determining call status, playing and recording voice messages, generating and detecting DTMF and tones, and controlling CT bus switching. Refer to the *Dialogic® NaturalAccess™ Software Developer's Manual* for a list of available NaturalAccess services.

Fusion

Fusion enables applications to perform media stream processing required for IVR tasks such as providing voice prompts, detecting and generating DTMF digits, or using speech recognition to query or deliver information.

Building a SIP application

To build a SIP application, use the SIP for NCC API and other NaturalAccess APIs. The following illustration shows the tasks a SIP application performs. Shaded tasks are optional.



The following table summarizes the tasks in the previous illustration:

Task	Application activity
Initialize NaturalAccess	Initializes NaturalAccess services and creates the required number of call control and media contexts.
Initialize connections to the SIP server	Invokes nccStartProtocol to establish a connection between the NCC SIP call control contexts and the SIP server.
Establish call	Uses NCC API functions to place outgoing calls, receive incoming calls, or both.
Establish media connection	Optional. Uses MSPP service functions to establish a media streaming channel.

Task	Application activity
Perform tasks	Uses functions from NaturalAccess or other APIs to perform tasks such as play or record voice, generate or detect DTMF tones, as well as send and receive faxes.
Disconnect call	Uses NaturalCallControl service functions to disconnect and release the call. The application can then establish another call.
Destroy media connection	Optional. Uses MSPP API functions to destroy the media streaming channel.

Implementing media streaming

When using Fusion software, you must configure the CG boards appropriately. For information about Fusion software, refer to the *Dialogic® NaturalAccess™ Fusion™ VoIP API Developer's Manual*.

4. Starting an SIP server

Configuring the SIP server

There are two aspects to configuring the SIP server.

- The first is the mode in which it is going to be run. For Windows, the software is run as either a Windows service or from the command line. For Linux and Solaris, the software can be run as either a daemon or from the command line.
- The second aspect of configuration is the programmable options that specify the behavior of the SIP stack.

Command line options

The following table describes the command line options that dictate how the SIP server is run:

Option	Description
register	Registers SIP as a Windows service.
unregister	Unregisters SIP as a Windows service.
start	Starts the SIP server as a background process.
stop	Stops the SIP server.
cmask	Changes the console mask of a running SIP server. Initially, this mask is set with the -v option.
fmask	Changes the log file mask of a running SIP server. Initially, this mask is set with the -V option.

The following table describes the command line options for the SIP server that are used when running in manual mode:

Option	Description
-h, -H, -?	Displays a list of options.
-f config_file	Specifies a configuration file name to override the defaults.
-d parm value	Sets parameters from the command line instead of from a configuration file.
-s	Reads the standard configuration file.
-L	Logs status information to a file.
-D	Prints configuration to stdout.

Option	Description																																						
-v mask_value	Specifies the status information displayed for debugging purposes. Specify mask_value by using a single hexadecimal value or by combining hexadecimal values using the OR operator. Use -h to display a list of options. This option is used in manual mode and causes logging to stdout. <table> <tr> <th>Mask</th><th>Description</th></tr> <tr> <td>0x00000001</td><td>Critical errors.</td></tr> <tr> <td>0x00000002</td><td>Not critical warnings.</td></tr> <tr> <td>0x00000004</td><td>General configuration information.</td></tr> <tr> <td>0x00000100</td><td>SIP events (incoming messages).</td></tr> <tr> <td>0x00000200</td><td>SIP commands (outgoing messages).</td></tr> <tr> <td>0x00000400</td><td>SIP state changes.</td></tr> <tr> <td>0x00000800</td><td>SIP addresses from messages.</td></tr> <tr> <td>0x00004000</td><td>SIP SDP from messages.</td></tr> <tr> <td>0x00008000</td><td>SIP REGISTER information.</td></tr> <tr> <td>0x00100000</td><td>NCC commands (API calls).</td></tr> <tr> <td>0x00200000</td><td>NCC events.</td></tr> <tr> <td>0x00400000</td><td>NCC state errors.</td></tr> <tr> <td>0x04000000</td><td>Miscellaneous debugging.</td></tr> <tr> <td>0x08000000</td><td>Licensing information.</td></tr> <tr> <td>0x10000000</td><td>SIP low-level subscription events.</td></tr> <tr> <td>0x20000000</td><td>SIP low-level transport events.</td></tr> <tr> <td>0x40000000</td><td>NCC low-level communication events.</td></tr> <tr> <td>0x80000000</td><td>Some debugging output.</td></tr> </table>	Mask	Description	0x00000001	Critical errors.	0x00000002	Not critical warnings.	0x00000004	General configuration information.	0x00000100	SIP events (incoming messages).	0x00000200	SIP commands (outgoing messages).	0x00000400	SIP state changes.	0x00000800	SIP addresses from messages.	0x00004000	SIP SDP from messages.	0x00008000	SIP REGISTER information.	0x00100000	NCC commands (API calls).	0x00200000	NCC events.	0x00400000	NCC state errors.	0x04000000	Miscellaneous debugging.	0x08000000	Licensing information.	0x10000000	SIP low-level subscription events.	0x20000000	SIP low-level transport events.	0x40000000	NCC low-level communication events.	0x80000000	Some debugging output.
Mask	Description																																						
0x00000001	Critical errors.																																						
0x00000002	Not critical warnings.																																						
0x00000004	General configuration information.																																						
0x00000100	SIP events (incoming messages).																																						
0x00000200	SIP commands (outgoing messages).																																						
0x00000400	SIP state changes.																																						
0x00000800	SIP addresses from messages.																																						
0x00004000	SIP SDP from messages.																																						
0x00008000	SIP REGISTER information.																																						
0x00100000	NCC commands (API calls).																																						
0x00200000	NCC events.																																						
0x00400000	NCC state errors.																																						
0x04000000	Miscellaneous debugging.																																						
0x08000000	Licensing information.																																						
0x10000000	SIP low-level subscription events.																																						
0x20000000	SIP low-level transport events.																																						
0x40000000	NCC low-level communication events.																																						
0x80000000	Some debugging output.																																						
-V mask_value	Same as lowercase -v except this option controls logging to the log file.																																						

Configuration parameters

The SIP stack configuration parameters are described in the following table. These parameters can be specified on the command line if you are running the SIP server manually, or in a configuration file when you run the SIP server as a service or daemon. If the SIP server is running as a service or a daemon, this configuration file must be named *nccsip.cfg* and must be placed in the *cfg* directory under the *oam* directory. The configuration file is an ASCII text file that contains a number of keywords and associated values. The keywords are organized into logical groups. The following table describes these keywords:

Keywords	Type	Default	Description
general.PRACKSupport	Bit mask	0x00	Type of PRACK support. Valid values: 0x00 - PRACK is not supported. 0x01 - PRACKs are used if supported header from the remote side indicated to use PRACKs. 0x02 - Set required header to use PRACK.

Keywords	Type	Default	Description
general.sendProtocolEvent	Boolean	False	Determines whether received SIP messages generate NCCEVN_PROTOCOL_EVENTS. Valid values: True - Generates NCCEVN_PROTOCOL_EVENT for all received SIP messages that have extra information, except those automatically handled by the SIP stack. The extra information is contained in the buffer attached to NCCEVN_PROTOCOL_EVENT. False - Only generates NCCEVN_PROTOCOL_EVENT for received SIP messages that have extra information, and that do not cause a change in call state. If an incoming message causes a change in call state and the general.sendDataInEvents keyword is set to a non-zero value, then extra information from the message is returned in the buffer of the associated NCC event. Note: If you are not sure about the NCC reported events, run the <i>ctasip</i> demo application.
general.manualAck	Boolean	False	Enables nccAcknowledgeCall to send an acknowledgement (ACK) in the sequence INVITE/200 OK/ACK. Valid values are: True - Application is responsible for sending ACK. False - SIP stack automatically generates ACK and nccAcknowledgeCall is unavailable.

Keywords	Type	Default	Description
general.manualPRACK	Boolean	False	Enables nccSendPrack and nccSendPrackResponse. Valid values: True - Application is responsible for sending PRACK or its acknowledgement. False - SIP stack automatically generates PRACK and the acknowledgement to PRACK. Note: general.PRACKSupport must not be 0 (zero) for this option to be valid.
general.numOutboundOnly	Integer	0	Number of outbound only channels.
general.numPorts	Integer	256	Specifies how many SIP ports the server supports.

Keywords	Type	Default	Description
general.sendDataInEvents	Bit mask	0	Specifies what data from SIP messages is sent inside NCC event buffers. Bit 0 (0x00000001) - Send the request URI from the INVITE message as the information element SIP_IE_REQUEST_URI. Bit 1 (0x00000002) - Send the SIP message body as the information element SIP_IE_BODY. Bit 2 (0x00000004) - Send unrecognized SIP message headers in the information element SIP_IE_OTHER_HEADER. Bit 3 (0x00000008) - Send the Call-ID header in the information element SIP_IE_CALL_ID. Bit 4 (0x00000010) - Send the CSeq header in the information element SIP_IE_CSEQ. Bit 5 (0x00000020) - Send the Content-Length header in the information element SIP_IE_CONTENT_LENGTH. Bit 6 (0x00000040) - Send the Content-Type header in the information element SIP_IE_CONTENT_TYPE. Bit 7 (0x00000080) - Send the To header in the SIP IE SIP_IE_TO_HEADER. Bit 8 (0x00000100) - Send the From header in the SIP IE SIP_IE_FROM_HEADER. Note: Setting general.sendDataInEvents to a non-zero value only has an effect when general.sendProtocolEvent is False.

Keywords	Type	Default	Description
general.maxRegClients	Integer	16	Specifies the number of simultaneous users that can be registered.
general.maxSubscriptions	Integer	256	Number of subscriptions the stack can simultaneously create. Subscriptions are used in the transfer process.
general.disconnectDelay	Integer	0	Amount of time in milliseconds to delay a disconnect event. This delay ensures that media stop events will come before a disconnect event is received.
general.rportInVia	Boolean	False	Indicates whether or not to add an rport parameter to the Via header.
general.noCtxtRejectCause	Integer	403	Response code that the SIP server will reply with if no more application contexts are available. This code is also sent if no applications are running. Valid range for this response is 400 through 699.
general.replaces	Boolean	False	Indicates if replaces header is supported. Must be enabled to have full functionality for nccTransferCall .
general.traceSizeLimitMb	Integer	0	Maximum size in megabytes for the SIP log. A value of 0 implies no limit on the size.
general.incomingReferSupport	Boolean	False	Specifies if an incoming REFER is automatically rejected by nmssip. False = Incoming REFER is rejected with a 501 Not Implemented message. True = Incoming REFER is presented to the application for processing.

Keywords	Type	Default	Description
log.file	String	OS specific	Directs status information to the specified directory. See Logging .
log.maskFile	Integer	0	Verbosity mask for logging to a file. See the -V option for <i>nmssip</i> .
log.maskConsole	Integer	0	Verbosity mask for logging to the console. See the -v option for <i>nmssip</i> .
options.sendAutoResponse	Boolean	False	Enabling/disabling automatic responses to OPTIONS methods.
options.allow	String		Parameter used in the OPTIONS automatic responses for the allow header, for example: options.allow = INVITE,ACK
options.accept	String		Parameter used in the OPTIONS automatic responses for the accept header, for example: options.accept = application/sdp
options.acceptEnc	String		Parameter used in the OPTIONS automatic responses for the acceptEnc header, for example: options.acceptEnc = gzip
options.language	String		Parameter used in the OPTIONS automatic responses for the language header, for example: options.language = en
options.supported	String		Parameter used in the OPTIONS automatic responses for the supported header, for example: options.supported = 100rel

Keywords	Type	Default	Description
outProxy.address	String		Address of the outbound proxy to use. If this address is set, the outProxy.hostName parameter is ignored.
outProxy.hostName	String		Hostname where the outbound proxy resides. For example, <i>sipproxy.company.com</i> .
outProxy.port	Integer	5060	Port for the outbound proxy to use.
outProxy.transport	String	UDP	Transport mechanism for the outbound proxy to use (either UDP or TCP).
tcp.address	String	0.0.0.0	TCP address that SIP uses.
tcp.port	Integer	5060	TCP port that SIP uses.
timer.cancelGeneralNoResponse	Integer	32000	Timeout for a response to a CANCEL request not including canceling an INVITE.
timer.cancelInviteNoResponse	Integer	32000	Timeout for a response to a CANCEL request sent after an INVITE.
timer.generalLinger	Integer	32000	Timer that the stack uses after sending a final response to be able to resend the response in case a retransmission of the initial request is received.
timer.generalRequestTimeout	Integer	32000	Timeout for a response to a general request.
timer.inviteLinger	Integer	32000	Timer that the stack uses after sending an ACK for an INVITE final response to resend the ACK in response to a retransmission of the OK response.

Keywords	Type	Default	Description
timer.provisional	Integer	180000	Timeout value to wait for a final response after a provisional response is received for an INVITE. A value of 0 (zero) means no timer is set.
timer.retransmissionT1	Integer	500	RFC 3261 defined timer.
timer.retransmissionT2	Integer	4000	RFC 3261 defined timer.
timer.retransmissionT4	Integer	5000	RFC 3261 defined timer.
udp.port	Integer	5060	UDP port that SIP uses.
udp.address	String	0.0.0.0	UDP address that SIP uses.

Sample configuration file

```
# Any line beginning with a # is a comment line
udp.port = 5062
general.PRACKSupport = 1
general.sendProtocolEvents = false
general.manualACK = true
general.manualPRACK = false
```

Running a SIP server

The SIP server can be run as a service or as a daemon depending on the installed operating system. In addition, the server can be run manually from the command line. Refer to [Configuring the SIP server](#) for the command line options.

Depending on the installed operating system, the SIP configuration file named *nccsip.cfg* must be located in one of the following directories:

Operating system	Directory
Windows	<i>nms\oam\cfg</i>
Linux or Solaris	<i>/opt/nms/oam/cfg</i>

To run the SIP server as a Windows service, register and run the server by entering the following commands:

```
nmssip register
nmssip start
```

For Linux and Solaris, run the SIP server by entering the following command:

```
nmssip start
```

The configuration file is read when the service or daemon is started. If the configuration parameters are changed, the service or daemon must be stopped and restarted.

Note: To change this behavior in Windows, use the Services menu under the Administrative Tools menu to change the startup type.

To unregister the SIP server under Windows, enter the following commands:

```
nmssip stop  
nmssip unregister
```

Logging

If logging is enabled, the output can be directed either to stdout or to a file. For Windows, the default location is *nms\oam\log* and for Linux and Solaris the default directory is */opt/nms/oam/log*.

If the configuration file contains the *log.file* keyword with a directory pathname, output is directed to the specified directory.

The *log.maskConsole* parameter is equivalent to the *-v* option; the *log.maskFile* parameter is equivalent to the *-V* option.

5. Establishing communication sessions

Exchanging SIP requests and responses

Dialogic implements SIP based on a client-server model. The client is a SIP application that communicates with the SIP server. The SIP server, *nmssip*, implements the SIP protocol stack, forming SIP messages as described in RFC 3261.

SIP automatically forms these messages when NaturalCallControl (NCC) functions are called. For voice over IP applications, the body of the SIP messages typically contains session description protocol (SDP) information describing the characteristics of the session.

Requests

The following table describes some of the SIP messages implemented by SIP:

Request	Description
INVITE	Invitation to establish a session. Contains a unique call identifier, destination address, and client address.
ACK	Acknowledgment by the initiating client that a server response was received.
BYE	Termination of a specific session or an attempted session.
PRACK	Acknowledgment of a provisional response.
CANCEL	Cancellation of a previous client INVITE request.
REGISTER	Registration in the domain server database of a user's current location. Users are identified by a SIP URI, for example, sip:user@dialogic.com or sip:user@1.1.1.1.

Responses

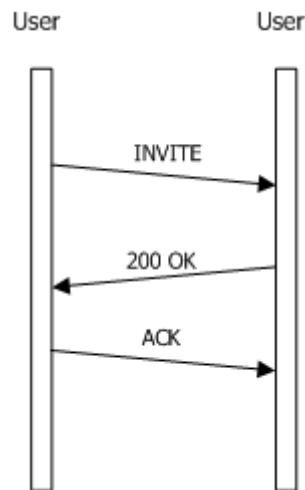
The response to a SIP request message is a SIP response. Response messages are numbered with codes grouped in ranges. The following table describes SIP responses:

Response code	Description
100 to 199	Provisional. Information about the progress of a request.
200 to 299	Success. A request successfully completed.
300 to 399	Redirection. Retry the request at another location.
400 to 499	Client error. Error in the client request.

Response code	Description
500 to 599	Server error. Error in the server response.
600 to 699	Global failure. Do not retry the request.

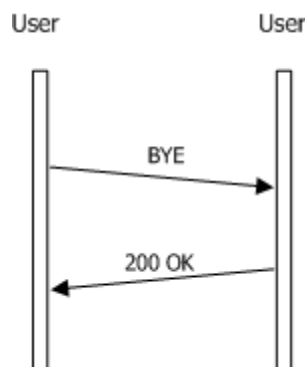
Establishing a session

The following illustration shows the sequence of requests and responses to establish a session:



Terminating a session

The following illustration shows the sequence of requests and responses to terminate a session:



Sending SIP messages

There are two ways to send a SIP message with the SIP for NCC API:

- Using [SIP for NCC API functions that send SIP messages automatically](#).
- Use **nccSendCallMessage** to send SIP messages that cannot be sent by using other NCC functions. When you use **nccSendCallMessage**, you must include values for SIP information elements.

Using SIP for NCC API functions that send SIP messages automatically

The following table lists the SIP for NCC API functions along with the SIP commands and responses they are associated with:

Function	SIP method	Response code
nccAcceptCall	INVITE	180
nccAcknowledgeCall	ACK	request
	INVITE	100
nccAnswerCall	INVITE	200
nccAutomaticTransfer	REFER	request
nccDisconnectCall	BYE or CANCEL	request
nccHandleTransferRequest	REFER	202 or 501
nccHoldCall	INVITE	request
nccPlaceCall	INVITE	request
nccRejectCall	INVITE	600, 603, 604, or 606
nccRetrieveCall	INVITE	request
nccSendCapabilityQuery	REQUEST	200
nccSendCapabilityResponse	OPTIONS	200
nccSendPrack	PRACK	request
nccSendPrackResponse	PRACK	200
nccTransferCall	REFER	request

For example, to send a SIP INVITE request when you are placing a call, invoke **nccPlaceCall**. The **nccPlaceCall** function generates a SIP INVITE request and sends it automatically.

Note: This method of sending a SIP message is compatible with SIP for NCC 1.0 functionality.

Using nccCallMessage

Use **nccSendCallMessage** to send SIP messages that cannot be sent using the NCC functions listed in the previous subsection. When you use **nccSendCallMessage**, include the following information elements with the function:

- [SIP_IE_BODY](#)
- [SIP_IE_CONTENT_LENGTH](#)
- [SIP_IE_CONTENT_TYPE](#)
- [SIP_IE_METHOD](#)
- [SIP_IE_OTHER_HEADER](#)
- [SIP_IE_RESPONSE_CODE](#)

The following table lists the SIP methods and response codes that you can use with **nccSendCallMessage**:

SIP method	Response codes	Notes
INVITE used to initiate a dialog	1xx	
INVITE issued in the context of a dialog (also called a re-INVITE)	request	Used in the connected state only.
	1xx – 6xx	Used in the connected state only.
	ACK	Used in the connected state only. The ACK message may be generated automatically, depending on the server configuration.

For example, to send a SIP INVITE request in the context of a dialog (a re-INVITE request), invoke **nccSendCallMessage** with the following information element values:

- SIP_IE_METHOD = INVITE
- SIP_IE_BODY = SDP offer for reINVITE

Receiving SIP messages

There are two ways for the SIP server to deliver the extra information associated with an incoming SIP request or response message:

- [Include the information in the buffer attached to the NCC event associated with the incoming message](#). This method is recommended.
- [Include the information in the buffer attached to NCCEVN_PROTOCOL_EVENT](#). This method is backwards compatible with version 1.0 of SIP for NCC API.

Including information in the buffer attached to an associated NCC event

With this delivery method, if a change in call state occurs as a result of the incoming SIP message, the SIP server delivers the extra information in the buffer attached to the associated NCC event. The NCC event uses the following information elements:

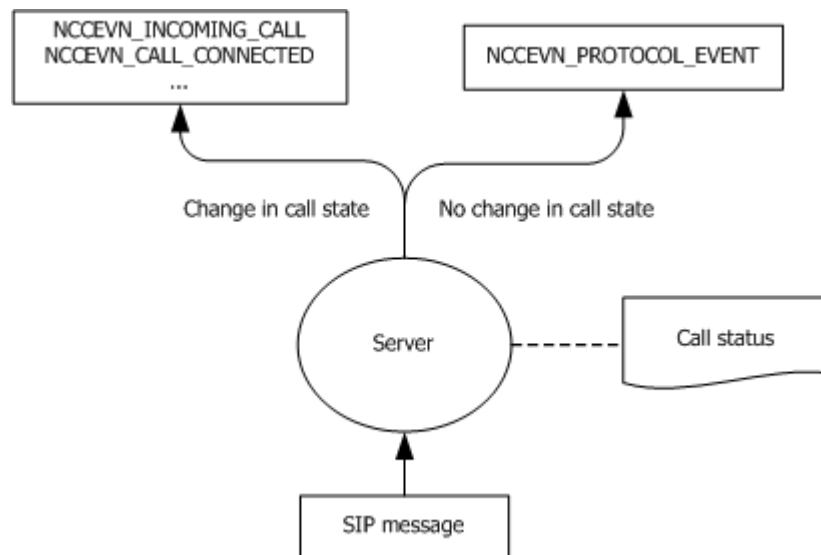
- [SIP_IE_BODY](#)
- [SIP_IE_CALL_ID](#)
- [SIP_IE_CONTENT_LENGTH](#)
- [SIP_IE_CONTENT_TYPE](#)
- [SIP_IE_CSEQ](#)
- [SIP_IE_METHOD](#)
- [SIP_IE_OTHER_HEADER](#)
- [SIP_IE_RESPONSE_CODE](#)

If no change in call state occurs as a result of the incoming SIP message, the SIP server delivers the extra information in the buffer attached to an [NCCEVN_PROTOCOL_EVENT](#). This event uses the information elements listed above.

For example, an incoming BYE message is returned to the application in the buffer of the NCCEVN_CALL_DISCONNECTED event, because the BYE message changes the call state. Because the re-INVITE message does not change the call state, an incoming SIP re-INVITE message is returned to the application in the buffer of NCCEVN_PROTOCOL_EVENT.

To configure the SIP server to use this method of delivering information, set the `general.sendProtocolEvent` keyword to `False` (the default), and the `general.sendDataInEvents` keyword to the desired bit mask. For more information, see [Configuration parameters](#).

The following illustration shows how the application receives SIP messages containing extra information:



Including information in the buffer attached to NCCEVN_PROTOCOL_EVENT

With this delivery method, the SIP server generates NCCEVN_PROTOCOL_EVENT every time it receives extra information in an incoming SIP message. The server puts the extra information in the buffer of NCCEVN_PROTOCOL_EVENT.

If the incoming SIP message triggers a call state change, the server generates two events: one for the call state change (such as NCCEVN_INCOMING_CALL), and one NCCEVN_PROTOCOL_EVENT.

NCCEVN_PROTOCOL_EVENT contains the following information elements:

- SIP_IE_CONTENT_TYPE
- SIP_IE_CONTENT_LENGTH
- SIP_IE_BODY
- SIP_IE_METHOD

To configure the server to generate NCCEVN_PROTOCOL_EVENTS for incoming messages that contain extra information, set `general.sendProtocolEvents` to True and `general.sendDataInEvents` to 0 (the default). For more information, see [Configuration parameters](#).

Receiving SIP events

All messages and events from the SIP server are returned to the application through the standard NaturalAccess event handling mechanism, along with standard NaturalAccess events and any events specific to NaturalAccess extensions. The SIP events are always NCC events that arrive in the CTA_EVENT data structure. For more information about NaturalAccess events, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

All events from...	Are prefixed with...
NCC API	NCCEVN_
ADI API	ADIEVN_
NaturalAccess	CTAEVN_

To receive these events, a NaturalAccess application can invoke **ctaWaitEvent**.

Some call-control-related NaturalAccess events are unsolicited. They can occur at any time, regardless of the application's current activities. Usually, unsolicited events indicate that something has happened on the trunk or in the session.

The eventmask parameter, which dictates whether certain informational call control events are generated, is found in the NCC_START_PARMS structure. For more information, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

6. SIP data and messages

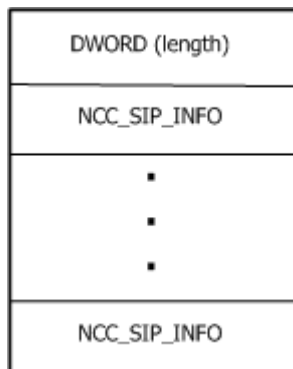
SIP for NCC API functions

NCC is a call control API that spans numerous telephony protocols. From a call control protocol perspective, the SIP for NCC API functions are a subset of the NCC functions. However, both the message body of SIP and the URI addresses in the protocol require a different treatment than those used in traditional telephony protocols.

The SIP for NCC API implementation makes use of the pointers to a buffer contained in the NCC function calls. The data in these buffers is function specific, but the general format is the same across those functions that accept these arguments.

Data structure formats

As shown in the following buffer illustration, the first word of the data structure is a DWORD that contains the length of data passed in the buffer. An array of NCC_SIP_INFO data structures follows the DWORD.



The definition of an NCC_SIP_INFO structure is as follows:

```
typedef struct
{
    WORD        tag;
    WORD        size;
    union
    {
        DWORD    number;
        BYTE     buffer[1];
    } d;
} NCC_SIP_INFO;
```

The first element of the structure is a tag that identifies the element type followed by the size and the data. The data element is either a DWORD or a data array; the data array is usually a string. The NCC_SIP_INFO structure specifies a SIP information element (IE).

Creating data structures

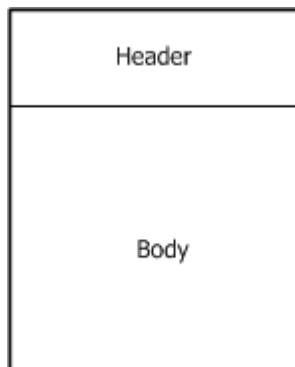
The *nccxsip.h* header file includes a set of macros for creating message buffers used by NCC API calls. To initialize the data buffer, use `NCC_SIP_INFO_INIT`. To add data, use `NCC_SIP_INFO_ADD_STR`, `NCC_SIP_INFO_ADD_NUM`, or `NCC_SIP_INFO_ADD_DATA`. The following example illustrates this procedure:

```
DWORD          b[256];                // data buffer
NCC_SIP_INFO    *ie;                  // work pointer
NCC_SIP_INFO_INIT ( b, &ie );         // initialize
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_TO_FULL_ADDRESS,
    _sip:calleduser@1.1.1.1:5060);     // add first SIP IE to buffer
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_FROM_FULL_ADDRESS,
    _sip:callinguser@1.1.1.2:5060);    // add second SIP IE to buffer
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING, 0 ); // add third SIP IE to buffer
```

This example contains some of the information that is passed to **nccPlaceCall**. The first two information elements (IEs) are the To (called party) and From (calling party) URLs. The IEs are copied to the SIP message header of an INVITE message that is the result of the NCC function call. The other piece of information is the session description protocol (SDP) encoding information. From an NCC perspective, this information is referred to as SIP information elements (SIP IEs).

SIP message formats

SIP is a text-based protocol. It uses SIP messages to send requests from a client to a server or to send responses from the server to the client. These messages use the following format:



Two methods are available to create the message body:

Method	Description
SIP IEs	Uses pre-defined SIP IEs to add session description protocol (SDP) information to the message. Refer to SIP information element values for a list of pre-defined values.
Transparent IEs	Formats the entire message body as a data array and includes that array as the <code>SIP_IE_BODY</code> element. For SDP information, this array is a string. The application must also supply the content-type header as a SIP IE so that the correct information appears in the SIP message. Refer to SIP information elements for a list. To add <code>SIP_IE_BODY</code> elements, use the <code>NCC_SIP_INFO_ADD_DATA</code> macro.

SIP information elements

The following table lists the SIP IEs that NCC uses:

Information element (IE)	Type	Description
To Header IEs		
SIP_IE_TO_ADDRESS	String	User address portion of the To header. To place a call, you can populate the address by setting the individual components or use SIP_IE_TO_FULL_ADDRESS.
SIP_IE_TO_HOST	String	IP address in the To header.
SIP_IE_TO_DISPLAY_NAME	String	Display name in the To header.
SIP_IE_TO_PORT	DWORD	Port number in the To header.
SIP_IE_TO_FULL_ADDRESS	String	Full SIP URI for the To header field. Use this IE to place a call.
SIP_IE_TO_URI_SCHEME	String	Scheme portion of the URI. For example, the scheme is tel in the example tel:+1234567.
SIP_IE_TO_URI_IDENTIFIER	String	Identifier portion of the URI. For example, the identifier is +1234567 in the example tel:+1234567.
From Header IEs		
SIP_IE_FROM_ADDRESS	String	User address portion of the From header.
SIP_IE_FROM_HOST	String	IP address in the From header.
SIP_IE_FROM_DISPLAY_NAME	String	Display name in the From header.
SIP_IE_FROM_PORT	DWORD	Port number in the From header.
SIP_IE_FROM_FULL_ADDRESS	String	Full SIP URI for the From header field. Use this IE to place a call.
SIP_IE_FROM_URI_SCHEME	String	Scheme portion of the URI. For example, the scheme is tel in the example tel:+1234567.

Information element (IE)	Type	Description
SIP_IE_FROM_URI_IDENTIFIER	String	Identifier portion of the URI. For example, the identifier is +1234567 in the example tel:+1234567.
Registration IEs		
SIP_IE_CONTACT_ADDRESS	String	Address at which the user can be contacted.
SIP_IE_REGISTRAR_ADDRESS	String	Address of the registrar.
SIP_IE_EXPIRES_HEADER	DWORD	Length of registration time in seconds.
SIP_IE_REGISTER_RENEW	DWORD	If this is a non-zero value, the SIP stack starts a timer and automatically renews the registration on the registrar before it expires.
Information for SIP message body using Transparent IEs		
SIP_IE_CONTENT_TYPE	String	Content-type header in the SIP message.
SIP_IE_BODY	Data	Message body.
SDP information for SIP message body using SIP IEs		
SIP_IE_SDP_PROTOCOL_VERSION	String	SDP protocol version.
SIP_IE_SDP_ORIGIN_USERNAME	String	User name in the origin field.
SIP_IE_SDP_ORIGIN_SESSION_ID	String	Session ID in the origin field.
SIP_IE_SDP_ORIGIN_VERSION	String	Version number in the origin field.
SIP_IE_SDP_ORIGIN_NETWORK_TYPE	String	IN
SIP_IE_SDP_ORIGIN_ADDRESS_TYPE	String	IP4
SIP_IE_SDP_ORIGIN_ADDRESS	String	Address in the origin field.
SIP_IE_SDP_SESSION_NAME	String	Session name.
SIP_IE_SDP_MEDIA_TYPE	DWORD	Media type. For valid values, refer to SIP_IE_SDP_MEDIA_TYPE .

Information element (IE)	Type	Description
SIP_IE_SDP_MEDIA_PORT	DWORD	Media port. For valid values, refer to SIP_IE_SDP_MEDIA_PORT .
SIP_IE_SDP_MEDIA_PROTOCOL	DWORD	Media protocol. For valid values, refer to SIP_IE_SDP_MEDIA_PROTOCOL .
SIP_IE_SDP_ENCODING	DWORD	Encoding. Use multiple entries if required. For valid values, refer to SIP_IE_SDP_ENCODING .
SIP_IE_SDP_CONNECTION_NET_TYPE	String	Network type: typically IN for Internet.
SIP_IE_SDP_CONNECTION_ADDR_TYPE	String	Address type: IP4
SIP_IE_SDP_CONNECTION_ADDRESS	String	Connection address: xxx.xxx.xxx.xxx
SIP_IE_SDP_ATTRIBUTE	String	Attribute.
Authentication IEs		
SIP_IE_AUTH_USER	String	User name.
SIP_IE_AUTH_PASSWORD	String	Password.
Other SIP IEs		
SIP_IE_SDP_ERROR	String	Error IE returned if incoming SDP contains an error.

Information element (IE)	Type	Description
SIP_IE_METHOD	String	IE that contains the SIP request or response message. For requests, SIP_IE_METHOD always contains a request name, such as INVITE or BYE. For responses, the contents of SIP_IE_METHOD differs depending on how the SIP server is configured. • If the SIP server is configured to deliver extra information in the buffer attached to the generated event, SIP_IE_METHOD contains a three digit response code such as 100. • If the SIP server is configured to deliver extra information in the buffer of NCC_PROTOCOL_EVENT, SIP_IE_METHOD contains the name of the request with which the response is associated. For information about configuring the SIP server, see Configuring the SIP server.
SIP_IE_OTHER_HEADER	String	IE that passes any user specified SIP headers to the SIP stack for inclusion in the SIP message.
SIP_IE_REQUEST_URI	String	URI from an incoming INVITE message.
SIP_IE_RESPONSE_CODE	Integer	IE that contains the response code associated with response messages: • For SIP requests, SIP_IE_RESPONSE_CODE contains a value of 0. • For response messages, SIP_IE_RESPONSE_CODE contains a three digit response code, such as 100. For more information, see Responses.
SIP_IE_CALL_ID	String	Call-ID header value.
SIP_IE_CSEQ	String	Command sequence header value.

Information element (IE)	Type	Description
SIP_IE_CONTENT_LENGTH	Integer	Length of SIP body.
SIP_IE_OPTIONS_HANDLE	DWORD	Options handle used to send the OPTIONS response (nccSendCapabilityResponse) or receive OPTIONS events.
SIP_IE_TRANSFER_TIMEOUT	Integer	User-supplied timeout in milliseconds for nccAutomaticTransfer .

SIP information element values

Some of the SIP IEs have predefined values. The following tables specify the SIP IE and list these values:

SIP_IE_SDP_MEDIA_TYPE

SIP IE	Decimal value
SDP_MEDIA_TYPE_AUDIO	1
SDP_MEDIA_TYPE_NAS	2
SDP_MEDIA_TYPE_VIDEO	3
SDP_MEDIA_TYPE_APP	4
SDP_MEDIA_TYPE_DATA	5
SDP_MEDIA_TYPE_IMAGE	6
SDP_MEDIA_TYPE_CONTROL	7

SIP_IE_SDP_MEDIA_PORT

SIP IE	Value
SIP_IE_SDP_MEDIA_PORT_ANY	1*
SIP_IE_SDP_MEDIA_PORT_NOTSET	2*
SIP_IE_SDP_MEDIA_PORT_IGNORE	3*
Actual port number	Numeric value

* These values do not correspond to port numbers.

SIP_IE_SDP_MEDIA_PROTOCOL

SIP IE	Decimal value
SDP_MEDIA_PROTOCOL_RTP	1
SDP_MEDIA_PROTOCOL_LOCAL	2
SDP_MEDIA_PROTOCOL_ATM	3
SDP_MEDIA_PROTOCOL_UDP	4
SDP_MEDIA_PROTOCOL_TCP	5

SIP_IE_SDP_ENCODING

For SIP_IE_ENCODING, see RFC 1890 for the assigned encodings. The following table lists some typical values:

Encoding method	Value
G711 mu-law	0
G711 A-law	8

SIP error codes

The following table lists the SIP-specific error codes returned with NCC events. These values are specified in the *nccxsip.h* header file.

Error	Hex	Description
NCC_SIP_PROTERR_MISSING_TO_ADDRESS	0x1C0101	To header information was not passed to the function.
NCC_SIP_PROTERR_MISSING_TO_HOST	0x1C0102	Internal error.
NCC_SIP_PROTERR_MISSING_FROM_ADDRESS	0x1C0103	From header information was not passed to the function.
NCC_SIP_PROTERR_MISSING_FROM_HOST	0x1C0104	Internal error.
NCC_SIP_PROTERR_PLACE_CALL_INIT	0x1C0105	Internal error.
NCC_SIP_PROTERR_PLACE_CALL	0x1C0106	Internal error.

Error	Hex	Description
NCC_SIP_PROTERR_CONTEXT_ERROR	0x1C0107	Internal error.
NCC_SIP_PROTERR_SEND_MSG_ERROR	0x1C0108	SIP message could not be sent.
NCC_SIP_PROTERR_MISSING_CONTACT_ADDRESS	0x1C0109	Contact header information was not passed to the function.
NCC_SIP_PROTERR_MISSING_REGISTRAR_ADDRESS	0x1C010A	URI of the registrar was not passed to the function.
NCC_SIP_PROTERR_SYNTAX_ERROR	0x1C010B	Syntax error exists in the fields passed to the function.
NCC_SIP_PROTERR_REGISTRATION_FAILED	0x1C010C	Registration failed.
NCC_SIP_PROTERR_TRANSFER_FAILED	0x1C010D	Transfer operation failed.
NCC_SIP_PROTERR_SUBSCRIPTION_FAILED	0x1C010E	Internal error; a SIP subscription request failed.
NCC_SIP_PROTERR_REFERER_FAILED	0x1C010F	Internal error; a REFER request failed during a transfer.
NCC_SIP_PROTERR_INITIALIZATION_FAILED	0x1C0110	Internal failure; occurred in an initialization sequence.
NCC_SIP_PROTERR_ILLEGAL_VALUE	0x1C0111	NCC function call failed because an illegal value was passed to the SIP stack.

Error	Hex	Description
NCC_SIP_PROTERR_BAD_HANDLE	0x1C0112	NCC function call failed because a bad handle was passed to the SIP stack (for the second call handle with nccTransferCall).

7. Using the NCC API

NCC API functions and SIP for NCC API

The NaturalCallControl (NCC) for SIP API implementation follows the NCC API call control model as described in the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*. This topic describes the NCC API from a SIP perspective.

The following table describes the NCC API functions that are supported in the SIP for NCC API and the associated SIP request or response messages. Functions that are specific to SIP for NCC API are also indicated.

NCC function	SIP request or response
nccAcceptCall	180 RINGING. Sends a provisional 180 Ringing response to an incoming INVITE request. For more information, refer to Accepting a call .
nccAcknowledgeCall	ACK. Sends ACK in the session establishment sequence INVITE/200 OK/ACK. This function is available only if the SIP stack is configured to operate in a manual acknowledgment mode. This function is SIP specific. The default mode of the stack is to send ACK automatically. For more information, refer to Acknowledging a call and to <code>general.manualACK</code> in Configuration parameters .
nccAnswerCall	200 OK. Sends a final 200 OK response to an incoming INVITE request. For more information, refer to Answering a call .
nccAutomaticTransfer	REFER. Instructs the party to whom you are connected to transfer the call, and causes a REFER to be sent to the remote party.
nccDisconnectCall	BYE. Sends a BYE response to terminate a session. If this function is called after nccPlaceCall , a CANCEL request is sent to the remote user in an attempt to cancel the establishment of a session. For more information, refer to Disconnecting a call .

NCC function	SIP request or response
nccGetCallStatus	No SIP response or request. This function retrieves the status of a specified call. For more information, refer to Getting call status information.
nccGetExtendedCallStatus	No SIP response or request. This function retrieves protocol-specific call status information that includes the To and From URIs and the session description protocol (SDP) information. For more information, refer to Getting extended call status information.
nccHandleTransferRequest	REFER. Allows the application to either accept (202 response) or reject (501 response) an incoming REFER request. If accepted, NCC automatically implements the transfer.
nccHoldCall	INVITE. Sends a RE-INVITE using the SDP used when establishing the call. The SDP is modified by marking the stream as sendonly or inactive if previously marked as sendrecv or recvonly respectively. For backward compatibility, the connection address is also set to 0.0.0.0.
nccPlaceCall	INVITE. Sends an INVITE request to initiate a session. For more information, refer to Placing a call.
nccQueryCapability	No SIP response or request. This function returns a capability mask that indicates the supported protocol capabilities. Refer to Capability mask for a list of supported capabilities.
nccRegisterUser	REGISTER. Sends a REGISTER request to register a user with a registrar. This function is SIP specific. For more information, refer to Registering a user.

NCC function	SIP request or response										
nccRejectCall	Rejects incoming calls with one of the following responses depending on the NCC rejection method: <table> <tr> <th>NCC method</th><th>SIP response</th></tr> <tr> <td>NCC_REJECT_PLAY_BUSY</td><td>600 Busy everywhere</td></tr> <tr> <td>NCC_REJECT_PLAY_RINGTONE</td><td>603 Decline</td></tr> <tr> <td>NCC_REJECT_PLAY_REORDER</td><td>604 Does not exist anywhere</td></tr> <tr> <td>NCC_REJECT_USER_AUDIO</td><td>606 Not acceptable</td></tr> </table> For more information, refer to Rejecting a call.	NCC method	SIP response	NCC_REJECT_PLAY_BUSY	600 Busy everywhere	NCC_REJECT_PLAY_RINGTONE	603 Decline	NCC_REJECT_PLAY_REORDER	604 Does not exist anywhere	NCC_REJECT_USER_AUDIO	606 Not acceptable
NCC method	SIP response										
NCC_REJECT_PLAY_BUSY	600 Busy everywhere										
NCC_REJECT_PLAY_RINGTONE	603 Decline										
NCC_REJECT_PLAY_REORDER	604 Does not exist anywhere										
NCC_REJECT_USER_AUDIO	606 Not acceptable										
nccReleaseCall	No SIP response or request. This function frees a SIP context so that the context can accept another inbound call or can place another outbound call. Even though the SIP server automatically acknowledges a BYE request to disconnect a call with a 200 OK response, nccReleaseCall is required to release the context.										
nccRetrieveCall	INVITE. Sends a RE-INVITE using the SDP used when establishing the call. The call must have been placed on hold prior to using this function.										
nccSendCallMessage	Sends messages that cannot be sent using other NCC API calls. When using nccSendCallMessage, decode extended parameters in the same way that you encode them for NCCEVN_PROTOCOL_EVENT. Note: The SIP server must be aware of the messages the user sends. Otherwise, nccSendCallMessage returns an error. For more information, see Sending SIP messages.										
nccSendCapabilityQuery	Requests capability information from the remote by sending a SIP OPTIONS message.										
nccSendCapabilityResponse	Responds with capability information to a SIP OPTIONS message.										

NCC function	SIP request or response
nccSendPrack	Sends a PRACK message. To use this function, the SIP stack must be configured to support PRACK manually. This function is SIP specific. For more information, refer to Sending a PRACK and to general.manualPRACK in the Configuration parameters table.
nccSendPrackResponse	Sends 200 OK in response to a PRACK. To use this function, the SIP stack must be configured to support PRACK manually. This function is SIP specific. For more information, refer to Acknowledging a PRACK and to general.manualPRACK in the Configuration parameters table.
nccStartProtocol	No SIP response or request. This function enables a context to run SIP. The trunk control program (TCP) name must be sip0.
nccStopProtocol	Frees the SIP context in the SIP server and stops it from running the SIP protocol. If a session exists on the context, a BYE request is sent to terminate the session.
nccTransferCall	REFER. If replaces header is enabled, a replaces header is built from the second call handle and placed in the REFER that is sent to the party on the current context. Otherwise, the REFER only contains the contact of the second call handle in the refer-to header.

Automatic responses

Some SIP requests elicit automatic responses. The following table summarizes the automatic responses:

Incoming SIP message	Automatic SIP message
INVITE	100 (TRYING)
200 (OK) following an INVITE request Note: general.manualACK can override this behavior.	ACK

Placing a call

When placing a call, use **nccPlaceCall** to send an INVITE request.

Prototype

DWORD **nccPlaceCall** (CTAHD *linehd*, char **calledaddr*, char **callingaddr*, void **mgrcallparms*, void **protcallparms*, NCC_CALLHD **callhd*)

Argument	Description
<i>linehd</i>	Line handle used to open the NCC service instance.
<i>calledaddr</i>	Not used, but cannot be NULL.
<i>callingaddr</i>	Not used.
<i>mgrcallparms</i>	Pointer to the SIP message information structure .
<i>protcallparms</i>	Not used.
<i>callhd</i>	Pointer to NCC_CALLHD. NCC assigns a valid call handle to a successfully initialized outbound call.

SIP message information structure

When an application calls **nccPlaceCall**, the SIP stack sends an INVITE. The NCC SIP message structure has two components:

- Message header
- Message body

Message header

The following table lists the required information elements (IEs) associated with the message header when **nccPlaceCall** is used:

Information element (IE)	Description
Message header information for SIP Request	
SIP_IE_TO_FULL_ADDRESS	SIP URI of the called user.
SIP_IE_FROM_FULL_ADDRESS	SIP URI of the calling user.

Message body

Two methods are available to create the message body:

Method	Description
Transparent IEs	Formats the entire message body as a string and includes the string as the SIP_IE_BODY element. The content-type header must be included so that the correct information appears in the SIP header. Refer to SIP information elements for a list.

Method	Description
SIP IEs	Uses pre-defined SIP IEs to add session description protocol (SDP) information to the message. Refer to SIP information element values for a list of pre-defined values.

SIP messages may contain multiple message bodies. Use the Transparent IE method to send a SIP message with multiple bodies. Precede each body being sent with a SIP_IE_CONTENT_TYPE header. Use the NCC_SIP_INFO_ADD_DATA macro to send binary bodies; for example, SIP-T, requires the use of binary bodies. When receiving a message with multiple bodies, a content length and content type IE precedes each received body IE.

You can also create a SIP message that does not have a body and includes only message headers. For example, the only IEs in a SIP INVITE request made using **nccPlaceCall** can be the To and From address IEs.

The following table lists the optional information elements (IEs) for the message body:

Information element (IE)	Description
Information for SIP message body using Transparent IEs	
SIP_IE_CONTENT_TYPE	Content-type header in the SIP message.
SIP_IE_BODY	Body of message.
SDP information for SIP message body using SIP IEs	
SIP_IE_SDP_ORIGIN_USERNAME	User name in the origin field.
SIP_IE_SDP_ORIGIN_SESSION_ID	Session ID in the origin field.
SIP_IE_SDP_ORIGIN_VERSION	Version number in the origin field.
SIP_IE_SDP_ORIGIN_ADDRESS	Address in the origin field.
SIP_IE_SDP_SESSION_NAME	Session name.
SIP_IE_SDP_CONNECTION_NET_TYPE	Network type: typically IN for Internet.
SIP_IE_SDP_CONNECTION_ADDR_TYPE	Address type: IP4
SIP_IE_SDP_CONNECTION_ADDRESS	Connection address in the following format: n.n.n.n where n.n.n.n is an IP address.
SIP_IE_SDP_MEDIA_TYPE	Media type. For valid values, refer to SIP_IE_SDP_MEDIA_TYPE .
SIP_IE_SDP_MEDIA_PORT	Media port. For valid values, refer to SIP_IE_SDP_MEDIA_PORT .

Information element (IE)	Description
SIP_IE_SDP_MEDIA_PROTOCOL	Media protocol. For valid values, refer to SIP_IE_SDP_MEDIA_PROTOCOL .
SIP_IE_SDP_ENCODING	Encoding. Use multiple entries if required. For valid values, refer to SIP_IE_SDP_ENCODING .

Although you can receive a message with SIP_IE_CONTENT_LENGTH, it is not necessary to include this IE when calling an NCC function that causes a SIP message to be sent. Specify the actual body length when passing body data using NCC_SIP_INFO_ADD_DATA.

Examples

The following coding example illustrates how to add SDP information using transparent IEs to a SIP IE buffer before calling **nccPlaceCall**:

```
// Add message SDP body using transparent IEs
//
DWORD b[256];
char bb[512];
char version[80], owner1[80], owner2[80], session[80], connection[80], time[80];
char media[80];

static char digits[40] = "0"

sprintf(version,      "v=0\r\n");
sprintf(owner1,       "o=user 01234567890 0987654321 IN IP4");
sprintf(owner2,       "sip:user@192.162.2.2:5060\r\n");
sprintf(session,      "s=NMS SIP\r\n");
sprintf(connection,   "c= IN IP4 192.168.2.2\r\n");
sprintf(time,         "t=0 0\r\n");
sprintf(media,        "m=audio 8006 RTP/AVP 0 8 3\r\n");
strcpy(bb, version);
strcat(bb, owner1);
strcat(bb, owner2);
strcat(bb, session);
strcat(bb, connection);
strcat(bb, time);
strcat(bb, media);
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_CONTENT_TYPE, "application/SDP" );
NCC_SIP_INFO_ADD_DATA ( b, &ie, SIP_IE_BODY, bb, strlen(bb) );

ret = nccPlaceCall( ctahd, digits, NULL, b, NULL, &callhd);
```

The following code example illustrates how to add SDP information using SIP IEs:

```
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_USERNAME, "user" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_SESSION_ID, "01234567890" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_VERSION, "0987654321" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_ADDRESS, "sip:user@192.162.2.2:5060");

NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_SESSION_NAME, "NMS SIP" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_NET_TYPE, "IN" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDR_TYPE, "IP4" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDRESS, "192.168.2.2" );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_TYPE, SDP_MEDIA_TYPE_AUDIO );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PORT, 8006 );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PROTOCOL, SDP_MEDIA_PROTOCOL_RTP );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING, 0 );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING, 8 );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING, 3 );

ret = nccPlaceCall( ctahd, digits, NULL, b, NULL, &callhd);
```

Answering a call

To answer an incoming call, use **nccAnswerCall**.

Prototype

DWORD NMSAPI **nccAnswerCall** (NCC_CALLHD *callhd*, unsigned *num_rings*, void **answerparms*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event.
<i>num_rings</i>	Not used.
<i>answerparms</i>	Pointer to a SIP information structure. The answerparms data structure is a buffer that contains SIP IEs. Typically, this buffer contains SDP information choosing the codec and specifying the IP address and port that the called party uses.

Example

```

DWORD          b[256];
NCC_SIP_INFO   *ie;

NCC_SIP_INFO_INIT ( b, &ie );

NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_USERNAME, "user" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_SESSION_ID, "01234567890" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_VERSION, "0987654321" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_ADDRESS, "<sip:user@192.162.2.2:5060>");

NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_SESSION_NAME, "NMS SIP" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_NET_TYPE, "IN" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDR_TYPE, "IP4" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDRESS, "192.168.2.2" );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_TYPE, SDP_MEDIA_TYPE_AUDIO );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PORT, 8006 );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PROTOCOL, SDP_MEDIA_PROTOCOL_RTP );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING, 0 );

ret = nccAnswerCall( callhd, nrings, b );

```

Accepting a call

To accept an incoming call, use **nccAcceptCall**. This function causes the provisional response *180 Ringing* to be sent to the calling party.

Prototype

DWORD NMSAPI **nccAcceptCall** (NCC_CALLHD *callhd*, unsigned *mode*, void **acceptparms*)

Argument	Description
callhd	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event.
mode	Not used.
acceptparms	Pointer to the acceptparms SIP information structure. The acceptparms data structure is a buffer that contains SIP IEs. This buffer can send SDP information. You are not required to send SDP information with every SIP message. If the SDP information is sent using nccAcceptCall , then nccAnswerCall can be passed a NULL pointer to the answerparms buffer.

Example

```
DWORD          b[256];
NCC_SIP_INFO   *ie;

NCC_SIP_INFO_INIT    ( b, &ie );

NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_USERNAME,  "user" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_SESSION_ID, "01234567890" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_VERSION,   "0987654321" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_ORIGIN_ADDRESS,    "<sip:user@192.162.2.2:5060>" );

NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_SESSION_NAME,      "NMS SIP" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_NET_TYPE, "IN" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDR_TYPE, "IP4" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_SDP_CONNECTION_ADDRESS,  "192.168.2.2" );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_TYPE,         SDP_MEDIA_TYPE_AUDIO );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PORT,         8006 );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_MEDIA_PROTOCOL,     SDP_MEDIA_PROTOCOL_RTP );

NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_SDP_ENCODING,           0 );

ret = nccAcceptCall( callhd, 0, b );
```

Rejecting a call

To reject an incoming call, use **nccRejectCall**.

Prototype

DWORD NMSAPI **nccRejectCall** (NCC_CALLHD *callhd*, unsigned *method*, void **rejectparms*)

Argument	Description										
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event.										
<i>method</i>	Method used to reject the call. <table> <tr> <th>Value</th><th>Resulting SIP message</th></tr> <tr> <td>NCC_REJECT_PLAY_REORDER</td><td>Sends a 604 response – Does not exist</td></tr> <tr> <td>NCC_REJECT_PLAY_RINGTONE</td><td>Sends a 603 response – Decline</td></tr> <tr> <td>NCC_REJECT_USER_AUDIO</td><td>Sends a 606 response – Not acceptable</td></tr> <tr> <td>NCC_REJECT_PLAY_BUSY</td><td>Sends a 600 response – Busy everywhere</td></tr> </table>	Value	Resulting SIP message	NCC_REJECT_PLAY_REORDER	Sends a 604 response – Does not exist	NCC_REJECT_PLAY_RINGTONE	Sends a 603 response – Decline	NCC_REJECT_USER_AUDIO	Sends a 606 response – Not acceptable	NCC_REJECT_PLAY_BUSY	Sends a 600 response – Busy everywhere
Value	Resulting SIP message										
NCC_REJECT_PLAY_REORDER	Sends a 604 response – Does not exist										
NCC_REJECT_PLAY_RINGTONE	Sends a 603 response – Decline										
NCC_REJECT_USER_AUDIO	Sends a 606 response – Not acceptable										
NCC_REJECT_PLAY_BUSY	Sends a 600 response – Busy everywhere										
<i>rejectparms</i>	Not used.										

Example

```
ret = nccRejectCall( callhd, reject, NULL );
```

Disconnecting a call

To disconnect a call, use **nccDisconnectCall**.

Prototype

DWORD NMSAPI **nccDisconnectCall** (NCC_CALLHD *callhd*, void **disconnectparms*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>disconnectparms</i>	Not used.

Example

```
ret = nccDisconnectCall( callhd, NULL );
```

Getting call status information

To retrieve call status information about a call, use **nccGetCallStatus**.

Prototype

DWORD NMSAPI **nccGetCallStatus** (NCC_CALLHD *callhd*, NCC_CALL_STATUS **callstatus*, unsigned *size*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .

Argument	Description																				
<i>callstatus</i>	Pointer to an address of a buffer to receive the call status information. This buffer must be at least the size of an NCC_CALL_STATUS structure. The following table lists the fields in the NCC_CALL_STATUS structure and indicates whether the fields are returned by SIP for NCC: <table> <tr> <th>Field</th><th>Returned</th></tr> <tr> <td>size</td><td>Yes</td></tr> <tr> <td>state</td><td>Yes</td></tr> <tr> <td>calledaddr</td><td>No</td></tr> <tr> <td>callingaddr</td><td>No</td></tr> <tr> <td>callingname</td><td>Yes</td></tr> <tr> <td>pendingcmd</td><td>Yes</td></tr> <tr> <td>held</td><td>No</td></tr> <tr> <td>direction</td><td>Yes</td></tr> <tr> <td>linehd</td><td>Yes</td></tr> </table>	Field	Returned	size	Yes	state	Yes	calledaddr	No	callingaddr	No	callingname	Yes	pendingcmd	Yes	held	No	direction	Yes	linehd	Yes
Field	Returned																				
size	Yes																				
state	Yes																				
calledaddr	No																				
callingaddr	No																				
callingname	Yes																				
pendingcmd	Yes																				
held	No																				
direction	Yes																				
linehd	Yes																				
<i>size</i>	Size of the buffer. This size should be set to the size of the NCC_CALL_STATUS buffer.																				

Example

```
NCC_CALL_STATUS info;
ret = nccGetCallStatus( callhd, &info, sizeof( NCC_CALL_STATUS ) );
```

Getting extended call status information

To retrieve extended call status information about a call, use **nccGetExtendedCallStatus**.

Prototype

DWORD NMSAPI **nccGetExtendedCallStatus** (NCC_CALLHD *callhd*, void **extendedcallstatus*, unsigned *size*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .

Argument	Description
<i>extendedcallstatus</i>	Pointer to an address of a buffer to receive the extended call status information.
<i>size</i>	Size of the buffer.

Details

extendedcallstatus is a buffer containing a fixed size structure that contains both data and offset information followed by a variable length data area containing SIP IEs. The NCC data structure `NCC_SIP_EXT_CALL_STATUS` specifies the layout of this buffer. It contains the To and From addresses for this session along with the SDP information for the remote and local users. A series of offsets is used to directly access SIP IEs stored in the data area. If the offset is zero, then the information element is not present.

```
typedef struct
{
    DWORD          size;      // size of extended call status information
    NCC_SIP_ADDRESS to;       // the to address or called user
    NCC_SIP_ADDRESS from;     // the from address or calling user
    NCC_SIP_SDP     sdpL;     // SDP information for local user
    NCC_SIP_SDP     sdpR;     // SDP information for remote user
    BYTE           data[1];   // SIP IEs (variable length data area)
};
```

Example

```
DWORD buffer[256];
DWORD toPort;
char toAddress[128];
NCC_SIP_EXT_CALL_STATUS *xcs = (NCC_SIP_EXT_CALL_STATUS *)buffer;
ret = nccGetExtendedCallStatus( callhd, xcs, sizeof(buffer) );

if (xcs->to.port)
toPort = xcs->to.port;
if (xcs->to.address)
strcpy(toAddress, NCC_SIP_STRING(*xcs,to.address));
```

Acknowledging a call

SIP uses a three way handshake in placing a call. The called party answers the INVITE command by sending a 200 OK response, and this in turn is acknowledged by the calling side by sending an ACK. Normally the SIP stack automatically sends the ACK response; however, an application can override this behavior by setting the nmssip server configuration keyword `generl.manualAck` to true. Once done, the application can manually send an ACK by using **`nccAcknowledgeCall`**. This function is useful if the application sends an empty INVITE. In this case, the called side proposes the codec information in the 200 OK message and the response by the calling side is in the ACK response.

Prototype

DWORD NMSAPI **`nccAcknowledgeCall`** (NCC_CALLHD *callhd*, void **acknowledgeparams*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>acknowledgeparms</i>	NULL or pointer to a buffer containing SIP IEs.

Example

```
ret = nccAcknowledgeCall( callhd, NULL );
```

Transferring a call

Use **nccAutomaticTransfer** or **nccTransferCall** to transfer a call.

Prototype

DWORD NMSAPI **nccAutomaticTransfer** (NCC_CALLHD ***callhd***, char ****digitstr***, unsigned ***transferwhen***, void ****transferparms***, void ****mgrcallparms***, void ****protcallparms***)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>Digitstr</i>	Not used with SIP for NCC.
<i>Transferwhen</i>	Specifies when to transfer the call. Valid values: • NCC_TRANSFER_PROCEEDING • NCC_TRANSFER_ALERTING • NCC_TRANSFER_CONNECTED
<i>Transferparms</i>	Not used with SIP for NCC.
<i>Mgrcallparms</i>	Not used with SIP for NCC.
<i>protcallparms</i>	Pointer to a buffer containing SIP IEs.

Details

nccAutomaticTransfer sends a REFER message to the transferee with the refer-to header containing the SIP_IE_CONTACT_ADDRESS provided in **protcallparms**. The stack disconnects the call when it receives:

- 2xx status for NCC_TRANSFER_PROCEEDING.
- 180 or 183 for NCC_TRANSFER_ALERTING.
- 200 for NCC_TRANSFER_CONNECTED.

Specify SIP_IE_TRANSFER_TIMEOUT to abort the transfer after **xxx** milliseconds have elapsed.

Example

```
DWORD      b[256];
NCC_SIP_INFO *ie;
static char digits[40] = "123";
static unsigned when = NCC_TRANSFER_PROCEEDING;
promptuns( "Transfer on: 1-proceeding, 2-alerting, 3-connected", &when );
promptstr( "Contact", sip.contact, sizeof(sip.contact) );

NCC_SIP_INFO_INIT ( b, &ie );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_CONTACT_ADDRESS, sip.contact ); // contact
ret = nccAutomaticTransfer( currentContext->callhd, digits, when, NULL, NULL, b );
```

Prototype

DWORD NMSAPI **nccTransferCall** (NCC_CALLHD *callhd1*, NCC_CALLHD *callhd2*, void **protcallparms*)

Argument	Description
<i>callhd1</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>callhd2</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>protcallparms</i>	Set this parameter to SIP_SUPERVISED_TRANSFER to enable supervised transfer with SIP for NCC.

Details

nccTransferCall performs a supervised call transfer using the two call handles specified when calling the function. The two calls must be in the NCC_CALLSTATE_CONNECTED before calling this function.

The function uses *callhd2* to construct a replaces header that is appended to a REFER message that is sent to the call specified by *callhd1*. In the REFER message, refer-to is set to the contact field from *callhd2* followed by the replaces header. On success, both calls are disconnected with NCC_DIS_TRANSFER for *callhd1* and NCC_DIS_SIGNAL for *callhd2*.

Example

```
ret = nccTransferCall(callhd1, callhd2, SIP_SUPERVISED_TRANSFER );
```

Placing on hold and retrieving a call

Use **nccHoldCall** or **nccRetrieveCall** to place a call on hold and retrieve a previously held call.

Prototype

DWORD NMSAPI **nccHoldCall** (NCC_CALLHD *callhd*, void **holdparms*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .

Argument	Description
<i>holdparms</i>	Not used with SIP for NCC.

Details

nccHoldCall sends a RE-INVITE using the SDP that was used when establishing the call. The SDP is modified by marking the stream as sendonly or inactive if previously marked as sendrecv or recvonly respectively. For backward compatibility, the connection address is also set to 0.0.0.0. The call must be in the connected state.

Example

```
ret = nccHoldCall(callhd, NULL);
```

Prototype

DWORD NMSAPI **nccRetrieveCall** (NCC_CALLHD *callhd*, void **holdparms*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>holdparms</i>	Not used with SIP for NCC.

Details

nccRetrieveCall sends a RE-INVITE using the SDP that was used when establishing the call. The call be on hold prior to using this function.

Example

```
ret = nccRetrieveCall(callhd, NULL);
```

Sending a PRACK

To send a PRACK, use **nccSendPrack**. PRACK is used as an acknowledgement of a provisional 1XX command. Using PRACK is a means for obtaining reliable delivery of provisional responses.

Prototype

DWORD NMSAPI **nccSendPrack** (NCC_CALLHD *callhd*, void **prackparms*)

Argument	Description
<i>callhd</i>	Call handle returned from nccPlaceCall .
<i>prackparms</i>	NULL or pointer to a buffer containing SIP IEs.

Details

To enable **nccSendPrack**, the nmssip server must be configured to support PRACK. The following configuration parameters must be set in the server to use **nccSendPrack**:

Parameter	Description
general.PRACKSupport	Set to 1 or 2.
general.manualPRACK	Set to a non zero value.

For more information, refer to [Configuration parameters](#).

Example

```
ret = nccSendPrack( callhd, NULL );
```

Acknowledging a PRACK

To acknowledge a PRACK, use **nccSendPrackResponse**.

Prototype

DWORD NMSAPI **nccSendPrackResponse** (NCC_CALLHD *callhd*, void **prackresponseparms*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event.
<i>prackresponseparms</i>	NULL or pointer to a buffer containing SIP IEs.

Details

In the SIP protocol, a PRACK is acknowledged by 200 OK message. A possible message sequence would be: the called side sends a 180 RINGING response, this is acknowledged by a PRACK, which in turn is acknowledged by a 200 OK response.

If the SIP stack is configured with **nccSendPrack** enabled, then **nccSendPrackResponse** is automatically enabled. If the application receives an NCCEVN_PRACK_INDICATION event, it must acknowledge the PRACK by calling **nccSendPrackResponse**.

Example

```
ret = nccSendPrackResponse( callhd, NULL );
```

Accepting or rejecting an incoming transfer request

An incoming transfer request is signaled by the receipt of a REFER. Use **nccHandleTransferRequest** to either accept or reject the incoming request. If the request is accepted, the SIP stack automatically attempts to transfer the call. NCC events are sent to the application detailing the progress of the transfer and whether the transfer succeeds or fails.

Prototype

DWORD NMSAPI **nccHandleTransferRequest** (NCC_CALLHD *callhd*, unsigned *accept*, void **sipIEs*)

Argument	Description
<i>callhd</i>	Call handle returned to the application with the NCCEVN_SEIZURE_DETECTED event or returned from nccPlaceCall .
<i>accept</i>	Flag indicating whether to accept or reject an incoming REFER. Non-zero = accept Zero = reject
<i>sipIEs</i>	NULL or pointer to a buffer containing SIP IEs.

Example

```
ret = nccHandleTransferRequest( callhd, true, sipIes );
```

Registering a user

To register a user with the registrar, use **nccRegisterUser**.

Prototype

DWORD NMSAPI **nccRegisterUser** (CTAHD *linehd*, void **sipIEs*)

Argument	Description
<i>linehd</i>	Line handle used to open the NCC service.
<i>sipIEs</i>	SIP information structure.

SIP registration information structure

The following table lists the optional or required information elements (IEs):

Information element (IE)	Required or optional	Description
SIP_IE_TO_ADDRESS	Required	Name of the user who is registering.
SIP_IE_FROM_ADDRESS	Optional	Name of the user who is requesting the registration. If this From address is omitted, the To address is used.
SIP_IE_CONTACT_ADDRESS	Required	Address where the user can be contacted.
SIP_IE_REGISTRAR_ADDRESS	Required	Address of the registrar.
SIP_IE_EXPIRES_HEADER	Optional	Length of time that the registration is to last (in seconds). The default time is 3600 seconds.
SIP_IE_REGISTER_RENEW	Optional	If non-zero, SIP stack will start a timer and automatically renew the registration on the registrar before it expires.

Example

```
NCC_SIP_INFO *ie;
NCC_SIP_INFO_INIT ( b, &ie );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_TO_ADDRESS, sip.to );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_CONTACT_ADDRESS, sip.contact );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_REGISTRAR_ADDRESS, sip.registrar );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_EXPIRES_HEADER, 3600 );
nccRegisterUser( ctahd_cc, b);
```

Requesting capability information

Use **nccSendCapabilityQuery** to send a SIP OPTIONS message when requesting capability information from the remote end. The **nccSendCapabilityQuery** parameters specify a line handle (**linehd**) and a SIP IE list to use for building the OPTIONS method headers (**sipIEs**).

Prototype

DWORD **nccSendCapabilityQuery**(CTAHD **linehd**, void * **sipIEs**)

Argument	Description
linehd	Line handle used to open the NCC service instance.
sipIEs	NULL or pointer to a buffer containing SIP IEs.

SIP capability information structure

The following table lists the optional and required information elements (IEs):

Information element (IE)	Required or optional	Description
SIP_IE_TO_FULL_ADDRESS	Required	SIP URI of the remote user.
SIP_IE_FROM_FULL_ADDRESS	Required	SIP URI of the local user.
SIP_IE_REQUEST_URI	Required	Address for requested URI.
SIP_IE_OTHER_HEADER	Required	Specifies the Accept header.
SIP_IE_CONTACT_ADDRESS	Optional	Contact address.

Example

The following example illustrates how to request capability information using **nccSendCapabilityQuery**:

```
DWORD          b[2048];
NCC_SIP_INFO   *ie;
NCC_SIP_INFO_INIT ( b, &ie );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_TO_FULL_ADDRESS, sip.to );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_FROM_FULL_ADDRESS, sip.from );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_REQUEST_URI, sip.requestUri );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER, "Accept:application/sdp" );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_CONTACT_ADDRESS, sip.contact );

ret = nccSendCapabilityQuery( ctahd, b );
```

Responding with capability information

The SIP stack can be configured to respond automatically to OPTIONS methods. You must customize several SIP OPTIONS headers in the SIP configuration file to do so. The SIP OPTIONS response is then formed according to those values. The sendAutoResponse configuration field must be set to enable this functionality.

The following example shows the OPTIONS keywords in the SIP Server configurable file:

```
options.sendAutoResponse = true
options.allow            = INVITE,ACK,BYE,REGISTER,REFER,NOTIFY,PRACK,CANCEL,SUBSCRIBE
options.accept           = application/sdp
options.acceptEnc        = gzip
options.language         = en
options.supported        = 100rel
```

The NCC application can also respond to the OPTIONS methods directly using the **nccSendCapabilityResponse** function. The parameters for this function specify a line handle (**linehd**) and a SIP IE list to build the OPTIONS method headers (**sipIEs**).

This function is called when receiving an NCCEVN_CAPABILITY_INDICATION event and only one SIP context should respond to the event. The NCCEVN_CAPABILITY_INDICATION event provides a SIP_IE_OPTIONS_HANDLE element that should be included in the **sipIEs** parameter of this function call.

Prototype

DWORD NMSAPI **nccSendCapabilityResponse**(CTAHD *linehd*, void **sipIEs*)

Argument	Description
<i>linehd</i>	Line handle used to open the NCC service instance.
<i>sipIEs</i>	NULL or pointer to a buffer containing SIP IEs.

SIP capability information structure

The following table lists the optional and required information elements (IEs):

Information element (IE)	Required or optional	Description
SIP_IE_OTHER_HEADER	Required	Use this header IE to specify response header values for the following header types: Allow, Accept, Accept-Encoding, Accept-Language and Supported.
SIP_IE_RESPONSE_CODE	Required	Response code to the received OPTIONS method.
SIP_IE_OPTIONS_HANDLE	Required	Options handle used to send the OPTIONS response.
SIP_IE_CONTACT_ADDRESS	Optional	Contact address.
SIP_IE_CONTENT_TYPE	Optional	Content-type header in the SIP message.
SIP_IE_BODY	Optional	Message Body.

Example

The following coding example illustrates how to respond with capability information using the **nccSendCapabilityResponse** function:

```

DWORD          b[2048];
NCC_SIP_INFO   *ie;
NCC_SIP_INFO_INIT ( b, &ie );
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_RESPONSE_CODE, 200 );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER,
"Allow:INVITE,ACK,BYE,REGISTER,REFER,NOTIFY,PRACK,CANCEL,SUBSCRIBE");
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER, "Accept:application/sdp");
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER, "Accept-Encoding:gzip");
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER, "Accept-Language:en");
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_OTHER_HEADER, "Supported:100rel");
NCC_SIP_INFO_ADD_NUM ( b, &ie, SIP_IE_OPTIONS_HANDLE, hOptions );
NCC_SIP_INFO_ADD_STR ( b, &ie, SIP_IE_CONTACT_ADDRESS, sip.contact );

ret = nccSendCapabilityResponse( ctahd, b);

```

NCC API events

When a SIP application performs call control functions or processes SIP requests and responses, the NCC API generates events. The names of these NCC events are defined in uppercase letters with an NCCEVN_ prefix.

CTA_EVENT structure

Events arrive in the form of the standard NaturalAccess event data structure.

```
typedef struct CTA_EVENT
{
    DWORD    id;                /* Event code (and source service id)    */
    CTAHD    ctahd;             /* Natural Access context handle         */
    DWORD    timestamp;         /* Timestamp                             */
    DWORD    userid;            /* User id (defined by ctaCreateContext) */
    DWORD    size;              /* Size of buffer if buffer != NULL      */
    void     *buffer;           /* Buffer pointer                         */
    DWORD    value;             /* Event status or event-specific data   */
    DWORD    objHd;             /* Service client side object handle     */
} CTA_EVENT;
```

The CTA_EVENT structure, returned by **ctaWaitEvent**, informs the application which event occurred on which context and includes additional information specific to the event.

The CTA_EVENT structure contains the following fields:

Field	Description
id	Contains an event code defined in the library header file. The event's prefix relates the event to a specific NaturalAccess library of functions. All NCC events are prefixed with NCCEVN_ . All NaturalAccess events are prefixed with CTAEVN_.
ctahd	Contains the context handle (line handle) returned from ctaCreateContext .
timestamp	Contains the time when the event was created in milliseconds.
userid	Contains the user-supplied id. This field is unaltered by Natural Access and facilitates asynchronous programming. Its purpose is to correlate a context with an application object or context when events occur.
size	Indicates the size (bytes) of the area pointed to by buffer. If the buffer is NULL, this field can be used to hold an event-specific value.
buffer	Points to data returned with the event. The field contains an application process address and the event's size field contains the actual size of the buffer.
value	Contains an event-specific reason code or error code.
objHd	Contains the call handle, if the event concerns a particular call.

The following table specifies standard NCC events that a SIP application can expect to receive:

NCC event	SIP method	Comments
NCCEVN_SEIZURE NCCEVN_INCOMING_CALL	INVITE	An incoming INVITE generates both of these events.
NCCEVN_CALL_PROCEEDING	100 TRYING	A 100 Trying response was received.
NCCEVN_REMOTE_ALERTING	180 RINGING	A 180 Ringing response was received.
NCCEVN_PROTOCOL_EVENT	All SIP requests and responses	If the SIP stack is configured to pass back all protocol events, all requests and responses are passed to the application through this event. The parameter that controls this behavior is <code>general.sendProtocolEvents</code> .
NCCEVN_CALL_CONNECTED	200 for INVITE or ACK	For outbound calls, a connected event results from an ACK being sent to the line. For inbound calls, receipt of an ACK generates this event.
NCCEVN_CALL_DISCONNECTED	BYE	A BYE request was received.
NCCEVN_CALL_HELD NCCEVN_CALL_RETRIEVED	A second INVITE request	To place a call on hold, a second INVITE request is sent to the user with the SDP connection address set to 0.0.0.0. The call is retrieved by another INVITE request that contains the original IP address.

NCC event	SIP method	Comments
NCCEVN_REGISTER_USER	OK in response to a REGISTER request	The NCCEVN_REGISTER_USER can also specify a failure to register the user. The value field contains SUCCESS if the register request succeeded. Otherwise, an error code is returned.
NCCEVN_PRACK_INDICATION	PRACK	Indicates the receipt of PRACK. It is received only if the stack is configured to operate in manual PRACK mode.
NCCEVN_PRACK_CONFIRMATION	200 OK or other final response	Indicates the receipt of a response to PRACK that an application sent manually. The value field contains the response code.
NCCEVN_PLACING_CALL	nccPlaceCall	An INVITE was successfully sent.
NCCEVN_CALL_RELEASED	Errors in outgoing INVITE request or response to nccReleaseCall	Indicates either a response to nccReleaseCall or termination of an outgoing call request due to errors in the INVITE request.
NCCEVN_ACCEPTING_CALL	nccAcceptCall	A 180 Ringing response was successfully sent.
NCCEVN_ANSWERING_CALL	nccAnswerCall	A 200 OK was successfully sent in response to an INVITE.
NCCEVN_REJECTING_CALL	nccRejectCall	Refer to NCC API functions and SIP for NCC API for the nccRejectCall 600 series response codes.
NCCEVN_START_PROTOCOL_DONE	nccStartProtocol	Indicates a response to nccStartProtocol .
NCCEVN_STOP_PROTOCOL_DONE	nccStopProtocol	Indicates a response to nccStopProtocol .

NCC event	SIP method	Comments
NCCEVN_PROTOCOL_ERROR	Typically sequence errors	The value field describes the specifics of the failure being reported.
NCCEVN_CAPABILITY_INDICATION	OPTIONS	Indicates the receipt of OPTIONS. This event is received only if the stack is configured to receive capability events (nccStartProtocol with the eventmask set with NCC_REPORT_CAPABILITY).
NCCEVN_CAPABILITY_RESPONSE	200 OK or other final response	The value field describes the specifics of the success or failure being reported for the OPTIONS method. This event is received only if the stack is configured to receive capability events (nccStartProtocol with the eventmask set with NCC_REPORT_CAPABILITY).

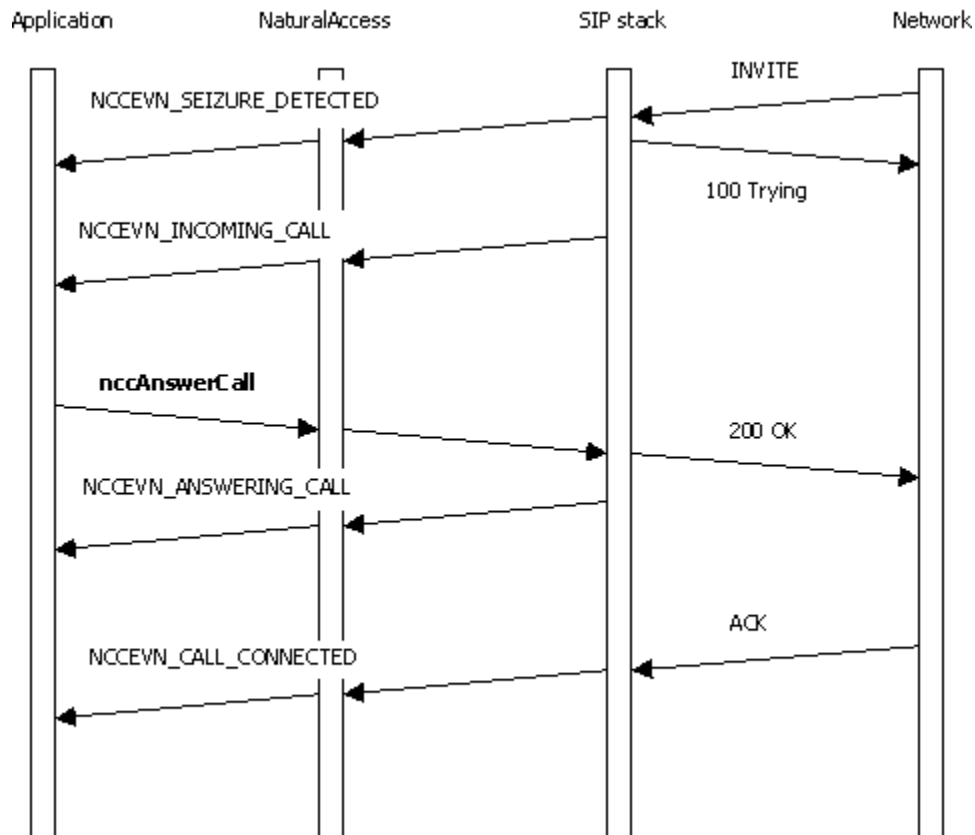
Sequence diagrams

This topic describes the exchange of commands and events between the network, SIP stack, NaturalAccess, and the application during various NCC API call control operations. The following function sequences are illustrated:

- [Inbound calls](#)
- [Outbound calls](#)
- [Local disconnect](#)
- [Remote disconnect](#)

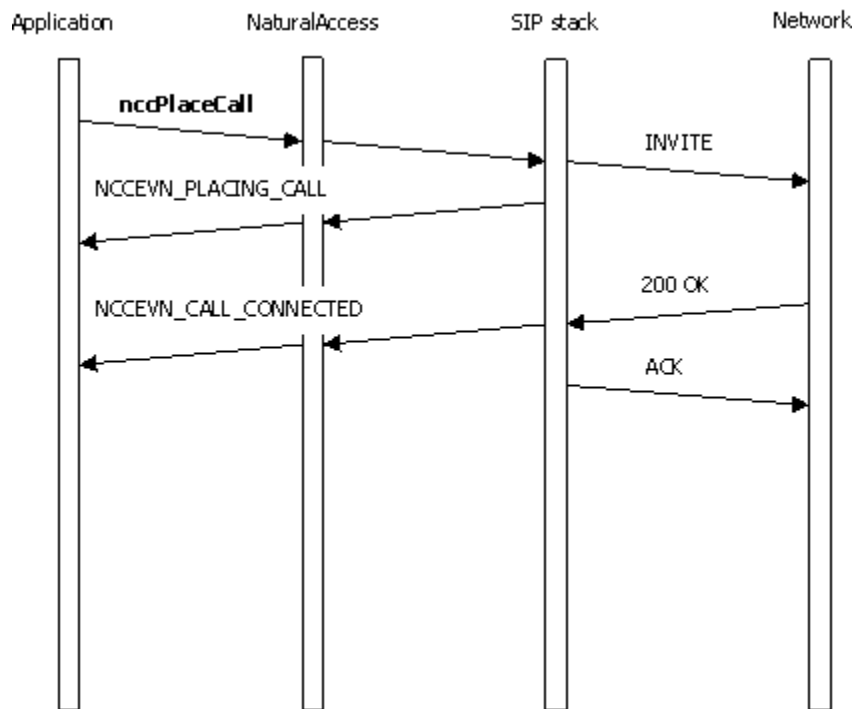
Inbound calls

The following illustration shows the function sequence for answering an inbound call.



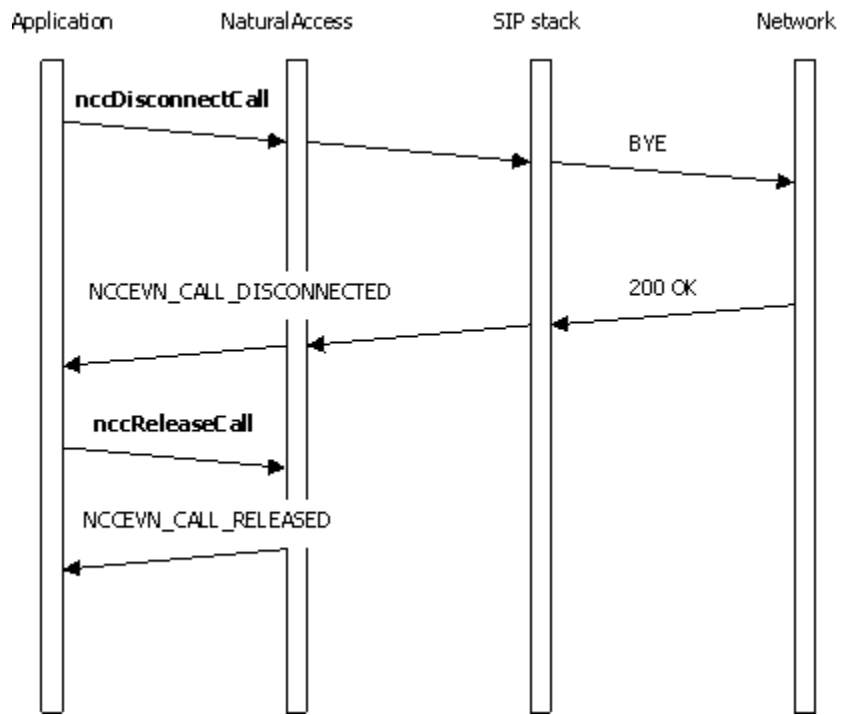
Outbound calls

The following illustration shows the function sequence for placing an outbound call:



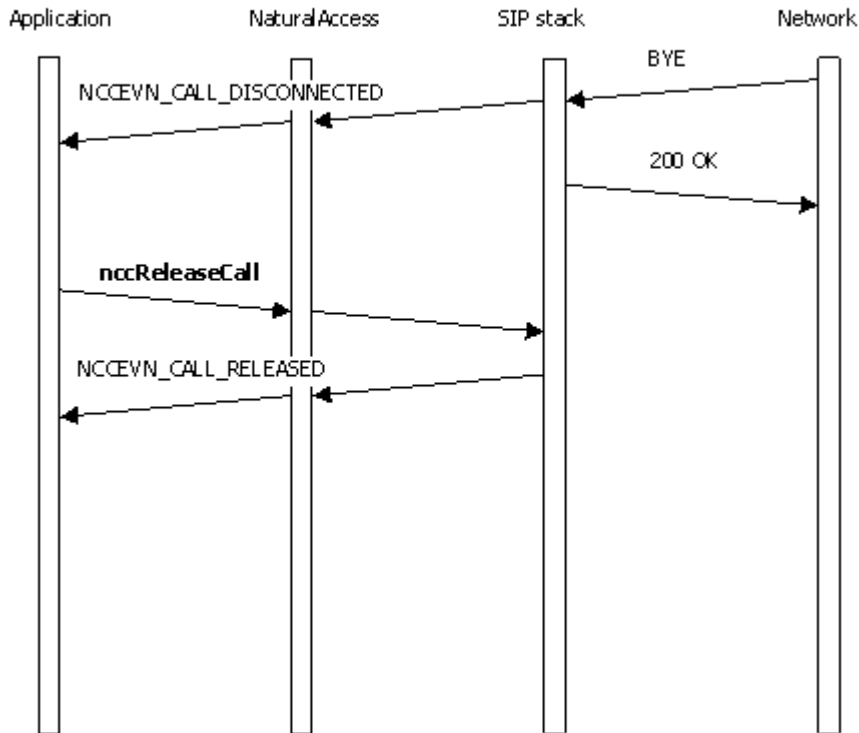
Local disconnect

The following illustration shows the command and event interchange for an application-initiated call release:



Remote disconnect

The following illustration shows the command and event interchange for a network-initiated call release:



Capability mask

SIP supports the **nccQueryCapability** function call. The capability mask indicates support of the following capabilities:

- NCC_CAP_ACCEPT_CALL
- NCC_CAP_EXTENDED_CALL_STATUS
- NCC_CAP_SEND_LINE_MESSAGE
- NCC_CAP_DISCONNECT_IN_ANY_STATE
- NCC_CAP_CALLER_ID
- NCC_CAP_REGISTER_USER
- NCC_CAP_ACKNOWLEDGE_CALL

Refer to the *NCC Service Developer's Reference Manual* for information about the first five mnemonic capabilities in the previous list. The SIP-specific mnemonic capabilities are described in the following table:

Mnemonic	If the bit is set...
NCC_CAP_REGISTER_USER	The SIP stack is capable of registering a user with a SIP registrar. For more information, refer to Registering a user .
NCC_CAP_ACKNOWLEDGE_CALL	The NCC API call nccAcknowledgeCall is enabled. For more information, refer to Acknowledging a call .

8. Using the MCC SDP library

Overview of the MCC SDP library

The MCC SDP library provides functions that describe session-level and media-level information for a media session. Session-level information includes the session name, session version, and originator of the session. Media-level information includes the media type, connection information, and codec information.

The MCC SDP library focuses on the parts of SDP applicable to the Offer/Answer model used by SIP. This model is defined by RFC 3264, *An Offer/Answer Model with Session Description Protocol (SDP)*.

You can use the MCC SDP library to do the following:

- [Read an SDP message](#)
- [Create an SDP message](#)
- [Print an SDP message](#)

The following table shows the SDP fields that the MCC SDP library enables you to create:

Abbreviation	Description
v	Protocol version field
o	Session owner and identifier field
s	Session name field
i	Session information field or media information field
c	Connection field
a	Session attribute field or media attribute field
m	Media description field

Note: There are no MCC SDP functions available for creating SDP session-level u, e, p, b, t, and k fields and media-level b and k fields.

Reading an SDP message

Use [mccReadSDP](#) to read an SDP message. For input, this function uses a memory buffer that contains an SDP message. For output, it generates a tree of C structures in a user-provided output buffer.

mccReadSDP returns a pointer to an [MCC_SDP](#) structure that contains session level information for the SDP message. The MCC_SDP structure points directly or indirectly to all other SDP structures.

The following example reads an SDP message:

```
MCC SDP *   sdp;
uint32_t    error;

char sdpData[1024];    /* Buffer to write SDP to */
char buffer[1024];     /* Buffer to create MCC_SDP structure in */

/* Read generated SDP into MCC_SDP */

sdp = mccdReadSDP( sdpData, mccdSdpGetLength(pc), buffer, sizeof(buffer), &error );

if ( sdp == 0 )
{
    printf("ERROR: failed to process SDP: %d\n", error);
    return;
}
```

Creating an SDP message

Create an SDP message by using MCC SDP functions to output SDP field values. To create an SDP message, perform these tasks:

Task	Action
1	Add session-level information to the SDP message.
2	Add one or more media descriptions to the SDP message.
3	Use the mccdSdpGetLength function to get the length of the generated SDP contents.

Adding session-level information to the SDP message

To add session-level information to an SDP message, follow these steps:

Step	Action	Associated function	Associated SDP field
1	Initialize the SDP context.	mccdSdpInit	N/A
2	Add a session description field.	mccdSdpAddSessionIP4 or mccSdpAddSessionIP6	v, o, s
3	Optionally, add an information field that applies to all media formats in the SDP message.	mccSdpAddInformation	i

Step	Action	Associated function	Associated SDP field
4	Optionally, add a session connection field that applies to all media formats in the SDP message. If you do not add a session connection field here, you must add one for each media type. For information, see Adding media descriptions to the SDP message .	mccdSdpAddConnIP4 or mccdSdpAddConnIP6	c
5	Optionally, add one or more general attributes that apply to all media formats in the session. Call mccdSdpAddAttribute for each session-level general attribute you want to add.	mccdSdpAddAttribute	a
6	Optionally, add one or more rtpmap attributes that apply to all media formats in the session. Call mccdSdpAddRtpmap for each session-level rtpmap attribute you want to add.	mccdSdpAddRtpmap	a

Note: There are no MCC SDP functions available for creating SDP session-level u, e, p, b, t, and k fields.

Adding media descriptions to the SDP message

To add one or more media descriptions to the SDP message, follow these steps:

Step	Action	Associated function	Associated SDP field
1	Begin a media description field.	mccdSdpAddMediaBegin	m
2	Add one or more media formats to the media description field. Call mccdSdpAddMediaFormat for each media type you want to add.	mccdSdpAddMediaFormat	m
3	Close the media description field.	mccdSdpAddMediaEnd	m
4	Optionally, add an information field that describes the media.	mccdSdpAddInformation	i

Step	Action	Associated function	Associated SDP field
5	Optionally, add a connection field for each specified media format. If you did not specify a session-level connection field, then this step is mandatory. Call mccdSdpAddConnIP4 or mccdSdpAddConnIP6 for each media-level connection field you want to add.	mccdSdpAddConnIP4 or mccdSdpAddConnIP6	c
6	Optionally, add one or more general attributes for each specified media format. Call mccdSdpAddAttribute for each media-level general attribute you want to add.	mccdSdpAddAttribute	a
7	Optionally, add one or more rtpmap attributes for each specified media format. Call mccdSdpAddRtpmap for each media-level rtpmap attribute you want to add.	mccdSdpAddRtpmap	a

Note: There are no MCC SDP functions available for creating SDP media-level b and k fields.

Creating an SDP message example

The following example creates an SDP message formatted for RTP/AVP and then obtains the length of the generated message:

```
MCC_SDP_CONTEXT_W context, *pc = &context;
MCC_SDP * sdp;
uint32_t error;

char sdpData[1024]; /* Buffer to write SDP to */
char buffer[1024]; /* Buffer to create MCC_SDP structure in */

/* Write SDP into sdpData */

mccdSdpInit          ( pc, sdpData, sizeof(sdpData) );
mccdSdpAddSessionIP4 ( pc, "user", 123, 124, "127.0.0.1" );
mccdSdpAddConnIP4    ( pc, "192.168.0.1" );
mccdSdpAddMediaBegin ( pc, MCCSDP_MEDIA_AUDIO, 8000, 0, "RTP/AVP" );
mccdSdpAddMediaFormat ( pc, "0" );
mccdSdpAddMediaFormat ( pc, "8" );
mccdSdpAddMediaEnd   ( pc );
mccdSdpAddInformation ( pc, "My audio stream" );
mccdSdpAddRtpmap      ( pc, "0", "PCMU", 8000, 0 );
mccdSdpAddRtpmap      ( pc, "8", "PCMA", 8000, 0 );

/* Print generated SDP */
printf ( " Generated SDP message: \n" );
printf ( "%.*s", mccdSdpGetLength(pc), sdpData );
```

Printing an SDP message

Use **mccdPrintSDP** to print the SDP message stored in the **MCC_SDP** structure. The **mccdPrintSDP** function uses a **printf**-like user-provided function to perform the printing. If you want to print to **STOUT**, you can use the standard C **printf** function as a parameter.

The following example creates an SDP message and then prints the entire SDP message from the **MCC_SDP_CONTEXT_W** context. For an example of printing specific SDP fields in an SDP message, see **mccdPrintSDP**.

```
MCC_SDP_CONTEXT_W context, *pc = &context;
_MCC_SDP * sdp;
uint32_t error;

char sdpData[1024]; /* Buffer to write SDP to */
char buffer[1024]; /* Buffer to create MCC_SDP structure in */

/* Write SDP into sdpData */

mccdSdpInit          ( pc, sdpData, sizeof(sdpData) );
mccdSdpAddSessionIP4 ( pc, "user", 123, 124, "127.0.0.1" );
mccdSdpAddConnIP4    ( pc, "192.168.0.1" );
mccdSdpAddMediaBegin ( pc, MCCSDP_MEDIA_AUDIO, 8000, 0, "RTP/AVP" );
mccdSdpAddMediaFormat ( pc, "0" );
mccdSdpAddMediaFormat ( pc, "8" );
mccdSdpAddMediaEnd   ( pc );
mccdSdpAddInformation ( pc, "My audio stream" );
mccdSdpAddRtpmap      ( pc, "0", "PCMU", 8000, 0 );
mccdSdpAddRtpmap      ( pc, "8", "PCMA", 8000, 0 );

/* Read generated SDP into MCC SDP */

sdp = mccdReadSDP( sdpData, mccdSdpGetLength(pc), buffer, sizeof(buffer), &error );

if ( sdp == 0 )
{
    printf("ERROR: failed to process SDP: %d\n", error);
    return;
}
/* Print read SDP */

printf("Generated SDP message:\n");
mccdPrintSDP( sdp, "", printf );
```

The example produces the following formatted SDP output:

```
Generated SDP message:
User name       : user
Session id      : 123
Session version : 124
Session name    : -
Origin          : IN, IP4, 127.0.0.1
Connection      : IN, IP4, 192.168.0.1
Media           : audio
  Port          : 8000 / 0
  Protocol      : RTP/AVP
  Format        : 0, PCMU, 8000
  Format        : 8, PCMA, 8000
Connection *    : IN, IP4, 192.168.0.1
```

9. MCC SDP functions

Using the MCC SDP function reference

The function reference describes each MCC SDP function and displays its prototype and return values. The function reference provides additional details, when necessary, and also provides a code sample for the function.

Prototype	The prototype is followed by a listing of the function's arguments. If a function uses a structure, the structure is listed as an argument.
Return values	A return value of MCCSDP_OK indicates that the function completed successfully. Other possible return values are listed in the description of each individual call.
Example	Example functions are taken from sample application programs shipped with the product. The notation <code>/* ... */</code> indicates additional code that is not shown.

mccdPrintSDP

Prints SDP information stored in the [MCC_SDP](#) structure.

Prototype

void **mccdPrintSDP** (MCC_SDP * *sdp*, const char * *prefix*, int *myprintf*(const char * *format*))

Argument	Description
<i>sdp</i>	Pointer to the MCC_SDP structure that contains the SDP message that you want to print.
<i>prefix</i>	(Optional) Pointer to the string to print at the beginning of every new line.
<i>myprintf</i>	A user-defined printf -like function to be used for printing. The standard C printf function may be used instead.
<i>format</i>	(Optional) Pointer to a string of SDP fields to print. Separate the variables with commas. If you do not specify an SDP field, mccdPrintSDP prints all SDP fields in the MCC_SDP structure.

Details

mccdPrintSDP uses a **printf**-like function to print a generated SDP message.

Examples

The following example prints all SDP fields in the [MCC_SDP](#) structure by using **mccPrintSDP** along with one call to a standard **printf** function:

```
MCC_SDP_CONTEXT_W context, *pc = &context;
MCC_SDP * sdp;
uint32_t error;

char sdpData[1024]; /* Buffer to write SDP to */
char buffer[1024]; /* Buffer to create MCC_SDP structure in */

/* Read generated SDP into MCC_SDP */

sdp = mccdReadSDP( sdpData, mccdSdpGetLength(pc), buffer, sizeof(buffer), &error );

if ( sdp == 0 )
{
    printf("ERROR: failed to process SDP: %d\n", error);
    return;
}
/* Print read SDP */

printf("Generated SDP message:\n");
mccPrintSDP( sdp, "", printf );
```

The following example shows an implementation of **mccPrintSDP**:

```
*/
void mccdPrintSDP ( MCC_SDP * sdp, const char * prefix,
                    int myprintf(const char * format, ...) )
{
    MCC_SDP_MEDIA * media;
    MCC_SDP_FORMAT * format;
    MCC_SDP_ATTRIBUTE * attr;

    myprintf("%sUser name : %s\n", prefix, sdp->username );
    myprintf("%sSession id : %" PRIu64 "\n", prefix, sdp->id );
    myprintf("%sSession version : %" PRIu64 "\n", prefix, sdp->version );
    myprintf("%sSession name : %s\n", prefix, sdp->name );
    myprintf("%sOrigin : %s, %s, %s\n", prefix,
              sdp->origin->nettype,
              sdp->origin->addrtype,
              sdp->origin->address );

    if ( sdp->connection )
    {
        myprintf("%sConnection : %s, %s, %s\n", prefix,
                  sdp->connection->nettype,
                  sdp->connection->addrtype,
                  sdp->connection->address );
    }

    for ( attr = sdp->attr; attr; attr = attr->next )
    {
        if ( attr->value )
            myprintf("%sAttribute : %s = %s\n", prefix, attr->name, attr->value );
        else
            myprintf("%sAttribute : %s\n", prefix, attr->name );
    }

    for ( media = sdp->media; media; media = media->next )
    {
        myprintf("%sMedia : %s\n", prefix, media->type );
        myprintf("%s Port : %u / %u\n", prefix, media->port, media->count );
        myprintf("%s Protocol : %s\n", prefix, media->protocol );

        for ( format = media->format; format; format = format->next )
        {
            myprintf("%s Format : ", prefix );
        }
    }
}
```

```

    if ( format->rtpmap == 0 )
        myprintf(" %s\n", format->id );
    else if ( format->rtpmap->parameter )
        myprintf(" %s, %s, %u, %s\n",
            format->id, format->rtpmap->encoding,
            format->rtpmap->clockrate, format->rtpmap->parameter );
    else
        myprintf(" %s, %s, %u\n",
            format->id, format->rtpmap->encoding,
            format->rtpmap->clockrate );
}

if ( media->connection )
{
    myprintf("%s    Connection    %c : %s, %s, %s\n", prefix,
        (media->connection == sdp->connection) ? '*' : ' ',
        media->connection->nettype,
        media->connection->addrtype,
        media->connection->address );
}

for ( attr = media->attr; attr; attr = attr->next )
{
    if ( attr->value )
        myprintf("%s    Attribute    : %s = %s\n", prefix, attr->name,
            attr->value );
    else
        myprintf("%s    Attribute    : %s\n", prefix, attr->name );
}
} // media
}

```

mccdReadSDP

Reads an SDP message from a user-provided memory buffer, and generates a tree of data structures containing information from the SDP message. The [MCC_SDP](#) data structure is the root data structure in the tree.

Prototype

MCC_SDP * **mccdReadSDP** (void * *sdpData*, unsigned *sdpSize*, void * *outData*, unsigned *outSize*, unsigned * *perror*)

Argument	Description
<i>sdpData</i>	Pointer to the input memory buffer.
<i>sdpSize</i>	Size of the input memory buffer.
<i>outData</i>	Pointer to the output memory buffer.
<i>outSize</i>	Size of the output memory buffer.
<i>perror</i>	Pointer to the returned completion code. Recognized values include: - MCCSDP_OK (Function completed normally.) - MCCSDP_MEMORY_ERROR - MCCSDP_SYNTAX_ERROR

Return values

mccReadSDP returns a pointer to the created [MCC_SDP](#) structure. If an error occurs during processing, the function returns NULL and places the error code into the variable pointed to by the ***perror*** argument.

Details

mccReadSDP does not use dynamic memory allocation for the generated data structures. Instead, it places all generated structures into the user-provided output memory buffer. This buffer must be large enough to contain all of the generated information.

The following error codes can occur:

Error code	Description
MCCSDP_MEMORY_ERROR	Output buffer is not large enough to hold the output.
MCCSDP_SYNTAX_ERROR	Output buffer format does not conform to RFC 2327.

Example

```
MCC_SDP *   sdp;
uint32_t    error;

char sdpData[1024];    /* Buffer to write SDP to */
char buffer[1024];     /* Buffer to create MCC SDP structure in */

/*Read generated SDP into MCC SDP */

sdp = mccdReadSDP( sdpData, mccdSdpGetLength(pc), buffer, sizeof(buffer), &error );

if ( sdp == 0 )
{
    printf("ERROR: failed to process SDP: %d\n", error);
    return;
}
```

mccdSdpAddAttribute

Adds a general attribute field (a=) to the output buffer.

Prototype

uint32_t **mccdSdpAddAttribute** (MCC_SDP_CONTEXT_W * ***context***, const char * ***name***, const char * ***value***)

Argument	Description
<i>context</i>	Pointer to the MCC_SDP_CONTEXT_W structure that stores the SDP message as you create its components.
<i>name</i>	Pointer to the attribute name.
<i>value</i>	Pointer to the attribute value.

Details

Depending on when you call **mccdSdpAddAttribute**, it adds either a session-level attribute or a media-level attribute to the SDP message. For more information, see [Creating an SDP message](#).

Return values

mccdSdpAddAttribute returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddAttribute ( pc, "T38FaxVersion", "0" );
```

creates the following SDP output:

```
a=T38FaxVersion:0
```

mccdSdpAddConnIP4

Adds the IPv4 address as a connection field (c) to the output buffer.

Prototype

```
uint32_t mccdSdpAddConnIP4 ( MCC_SDP_CONTEXT_W * context, const char * address)
```

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.
address	Pointer to the origin IPV4 address, which is formatted as a string.

Return values

mccdSdpAddConnIP4 returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddConnIP4 ( pc, "192.168.0.1" );
```

creates the following SDP output:

```
c=IN IP4 192.168.0.1
```

mccdSdpAddConnIP6

Adds the IPV6 address as a connection field (c) to the output buffer.

Prototype

```
uint32_t mccdSdpAddConnIP6 ( MCC_SDP_CONTEXT_W * context, const char * address)
```

Argument	Description
<i>context</i>	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.
<i>address</i>	Pointer to the origin IPV6 address, which is formatted as a string.

Return values

mccdSdpAddConnIP6 returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddConnIP6 ( pc, "fe80::1" );
```

creates the following SDP output:

```
c=IN IP6 fe80::1
```

mccdSdpAddInformation

Writes the SDP information field (i) to the output buffer.

Prototype

```
uint32_t mccdSdpAddInformation ( MCC_SDP_CONTEXT_W * context, const char * info)
```

Argument	Description
<i>context</i>	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.
<i>info</i>	Pointer to SDP session information.

Details

Depending on when you call **mccdSdpAddInformation**, it adds a session-level information field or a media-level information field to the SDP message. For more information, see [Creating an SDP message](#).

Return values

mccdSdpAddInformation returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddInformation ( pc, "My audio stream" );
```

creates the following SDP output:

```
i=My audio stream
```

mccdSdpAddMediaBegin

Adds the beginning of the media description field (m) to the output buffer.

Prototype

```
uint32_t mccdSdpAddMediaBegin ( MCC_SDP_CONTEXT_W * context, const char * media_s, uint32_t port, uint32_t count, const char * proto_s)
```

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.
media_s	Pointer to the media type, which is formatted as a string constant. Recognized values include: • MCCSDP_MEDIA_APP • MCCSDP_MEDIA_AUDIO • MCCSDP_MEDIA_CONTROL • MCCSDP_MEDIA_DATA, • MCCSDP_MEDIA_IMAGE • MCCSDP_MEDIA_VIDEO
port	Media port.
count	Media port count. A value of 0 indicates no port count.
proto_s	Pointer to the media protocol, which is formatted as a string constant. Recognized values include: • MCCSDP_PROT_RTPAVP • MCCSDP_PROT_UDP • MCCSDP_PROT_TCP • MCCSDP_PROT_UDPTL

Return values

mccdSdpAddMediaBegin returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Details

After calling **mccdSdpAddMediaBegin**, call **mccdSdpAddMediaFormat** one or more times to add media formats, and then **mccdSdpAddMediaEnd** to terminate the media field. For more information, see [Creating an SDP message](#).

Example

This example:

```
mccdSdpAddMediaBegin ( pc, MCCSDP_MEDIA_AUDIO, 8000, 0, "RTP/AVP" );
```

creates the following SDP output:

```
m=audio 8000 RTP/AVP
```

mccdSdpAddMediaEnd

Indicates the ending of the media description field (m).

Prototype

```
uint32_t mccdSdpAddMediaEnd ( MCC_SDP_CONTEXT_W * context)
```

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.

Details

Call **mccdSdpAddMediaEnd** after using **mccdSdpAddMediaBegin** and **mccdSdpAddMediaFormat**. For more information, see [Creating an SDP message](#).

Return values

mccdSdpAddMediaEnd returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddMediaEnd (pc );
```

terminates the sample media description field created by **mccdSdpAddMediaBegin** and **mccdSdpAddMediaFormat**.

mccdSdpAddMediaFormat

Adds a media format to the media description field (m) in the output buffer.

Prototype

```
uint32_t mccdSdpAddMediaFormat ( MCC_SDP_CONTEXT_W * context, const char * format)
```

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores the SDP message as you create its components.
format	Pointer to the media format payload identifier, which is formatted a string constant. For audio and video media, use the media payload type as defined in the RTP Audio/Video profile (RFC 1890).

Details

Call **mccdSdpAddMediaFormat** once for each media format you want to add. A list of formats implies that all of these formats may be used in the session. The first format in the list is the default format for the session.

Return values

mccdSdpAddMediaFormat returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddMediaFormat ( pc, "0" );
mccdSdpAddMediaFormat ( pc, "8" );
```

adds media formats 0 and 8 to the SDP m field started by **mccdSdpAddMediaBegin**.

```
m=audio 8000 RTP/AVP 0 8
```

mccdSdpAddRtpmap

Adds the rtpmap attribute field (a=rtpmap) to the output buffer.

Prototype

```
uint32_t mccdSdpAddRtpmap ( MCC_SDP_CONTEXT_W * context, const char * format,
const char * encoding, uint32_t clockrate, const char * param)
```

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores the SDP message as you create its components.
format	Pointer to the media format payload identifier, which is formatted as a string constant. For audio and video media, use the media payload type as defined in the RTP Audio/Video profile (RFC 1890).
encoding	Pointer to the RTP encoding.
clockrate	RTP clock rate.
param	Pointer to an optional parameter. A value of 0 indicates no optional parameter.

Details

Depending on when you call **mccdSdpAddRtpmap**, it adds either a session-level rtpmap attribute or a media-level rtpmap attribute to the SDP message. For more information, see [Creating an SDP message](#).

Return values

mccdSdpAddRtpmap returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddRtpmap ( pc, "0", "PCMU", 8000, 0 );
```

creates the following SDP output:

```
a=rtpmap:0 PCMU/8000
```

mccdSdpAddSessionIP4

Writes the mandatory session description fields v, o, and s to the output buffer, using the IPV4 address for origin field o.

Prototype

uint32_t **mccdSdpAddSessionIP4** (MCC_SDP_CONTEXT_W * **context**, const char * **username**, uint32_t **id**, uint32_t **version**, const char * **addr**)

Argument	Description
context	Pointer to the MCC_SDP_CONTEXT_W structure that stores the SDP message as you create its components.
username	Pointer to the origin user name.
id	Session identifier.
version	Session version.
addr	Pointer to the IPV4 address.

Return values

mccdSdpAddSessionIP4 returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddSessionIP4 ( pc, "user", 123, 124, "127.0.0.1" );
```

creates the following SDP output:

```
v=0
o=user 123 124 IN IP4 127.0.0.1
s=-
```

mccdSdpAddSessionIP6

Writes the mandatory session description fields v, o, and s to the output buffer, using the IPV6 address for origin field o.

Prototype

uint32_t **mccdSdpAddSessionIP6** (MCC_SDP_CONTEXT_W * **context**, const char * **username**, uint32_t **id**, uint32_t **version**, const char * **addr**)

Argument	Description
<i>context</i>	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.
<i>username</i>	Pointer to the origin user name.
<i>id</i>	Session identifier.
<i>version</i>	Session version.
<i>addr</i>	Pointer to the origin IPV6 address.

Return values

mccdSdpAddSessionIP6 returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

This example:

```
mccdSdpAddSessionIP6 ( pc, "user", "fe80::1" );
```

creates the following SDP output:

```
v=0
o=user 123 124 IN IP6 fe80::1)
s=-
```

mccdSdpGetLength

Returns the length of the generated SDP contents.

Prototype

uint32_t **mccdSdpGetLength** (MCC_SDP_CONTEXT_W * ***context***)

Argument	Description
<i>context</i>	Pointer to the MCC_SDP_CONTEXT_W structure that stores an SDP message as you create its components.

Example

```
mccdSdpGetLength ( pc );
```

mccdSdpInit

Initializes the SDP writing context for writing to the specified memory buffer.

Prototype

```
uint32_t mccdSdpInit ( MCC_SDP_CONTEXT_W * context, void * data, uint32_t size)
```

Argument	Description
context	Pointer to the context to initialize.
data	Pointer to the output memory buffer.
size	Size of the output memory buffer.

Return values

mccdSdpInit returns either MCCSDP_OK (function completed successfully) or MCCSDP_MEMORY_ERROR.

Example

The following example sets a 1024 byte memory buffer (sdpData) as an output buffer for the MCC_SDP_CONTEXT_W context pointed to by pc:

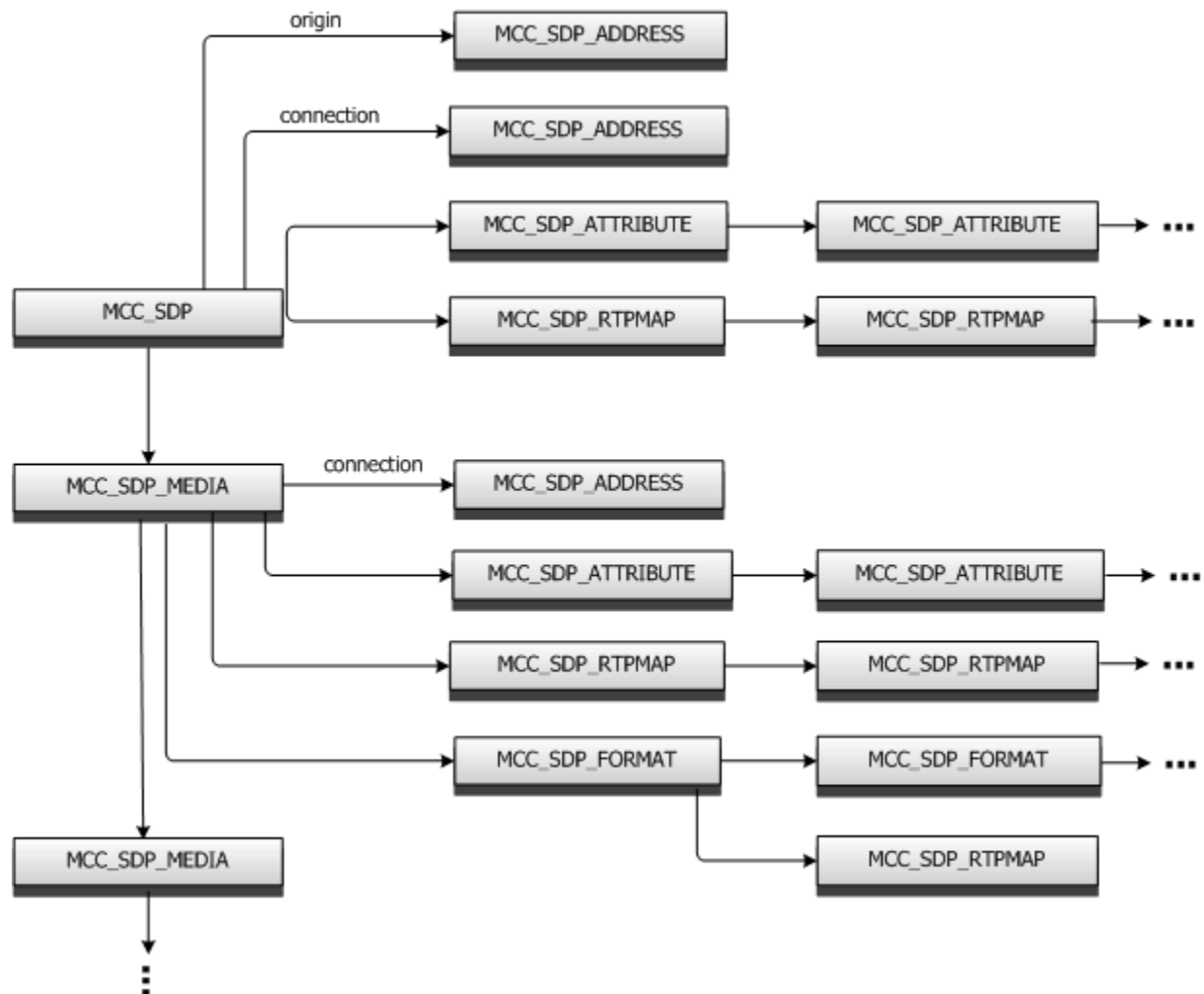
```
char sdpData[1024];
mccdSdpInit ( pc, sdpData, sizeof(sdpData) );
```

10. MCC SDP structures

Using the MCC SDP structure reference

This section provides an alphabetical reference to the MCC SDP structures used in the NCC service. The topics in the structure reference include the structure definition and table of field descriptions.

The following illustration shows the structure hierarchy in the MCC SDP library:



MCC_SDP

Represents an SDP message. The MCC_SDP structure contains session level information for the message, and pointers to other SDP structures that further describe the message.

MCC_SDP is the only structure returned directly by [mccReadSDP](#).

Definition

```
typedef struct MCC_SDP
{
    const char *      info;
    const char *      username;
    const char *      name;
    uint32_t          id;
    uint32_t          version;
    MCC_SDP_ADDRESS *  origin;
    MCC_SDP_ADDRESS *  connection;
    MCC_SDP_ATTRIBUTE * attr;
    MCC_SDP_RTPMAP *   rtpmap;
    MCC_SDP_MEDIA *    media;
} MCC_SDP;
```

Fields

Field	Type	Description	SDP field	Required/optional
info	const char	Information about the session.	i	Optional
username	const char	Session origin user name.	o	Required
name	const char	Session name.	s	Required
id	uint32_t	Session identifier.	o	Required
version	uint_t	Session version.	o	Required
origin	MCC_SDP_ADDRESS	Session origin address.	o	Required

Field	Type	Description	SDP field	Required/optional
connection	MCC_SDP_ADDRESS	Pointer to the session-level connection address as defined by the MCC_SDP_ADDRESS structure. The value of this field is 0 if there is no session-level connection address associated with this SDP message. If a session-level connection address is not present, then media-level connection addresses must be present.	c	Optional
attr	MCC_SDP_ATTRIBUTE	Pointer to the first session-level general attribute as defined by the MCC_SDP_ATTRIBUTE structure. The value of this field is 0 if there are no session-level general attributes associated with this SDP message.	a	Optional
rtpmap	MCC_SDP_RTPMAP	Pointer to the first session-level rtpmap attribute as defined by the MCC_SDP_RTPMAP structure. The value of this field is 0 if there are no session-level rtpmap attributes in this SDP message.	a	Optional
media	MCC_SDP_MEDIA	Pointer to the first media description as defined by the MCC_SDP_MEDIA structure. The value of this field is 0 if there are no media descriptions in this SDP message.	m	Optional

MCC_SDP_ADDRESS

Represents an SDP origin address or connection address.

Definition

```
typedef struct MCC_SDP_ADDRESS
{
    const char *    nettype;
    const char *    addrtype;
    const char *    address;
} MCC_SDP_ADDRESS;
```

Fields

Field	Type	Description	SDP field	Required/optional
nettype	const char	Network type. Recognized value is MCCSDP_NETTYPE_IN (Internet address).	o or c	Required
addrtype	const char	Address type. Recognized values are: - MCCSDP_ADDRATYPE_IP4 - MCCSDP_ADDRATYPE_IP6	o or c	Required
address	const char	Destination IP address.	o or c	Required

MCC_SDP_ATTRIBUTE

Represents a session-level general attribute or media-level general attribute in an SDP message. MCC_SDP_ATTRIBUTE represents:

- A session-level general attribute, when it is used as a parameter in the [MCC_SDP](#) structure.
- A media-level general attribute, when it is used as a parameter in the [MCC_SDP_MEDIA](#) structure.

When an SDP message contains multiple attributes of the same type (session-level or media-level), they are represented by a linked list of MCC_SDP_ATTRIBUTE structures.

Note: Rtpmap attributes are represented by the [MCC_SDP_RTPMAP](#) structure instead of the MCC_SDP_ATTRIBUTE structure.

Definition

```
typedef struct _MCC_SDP_ATTRIBUTE
{
    struct _MCC_SDP_ATTRIBUTE * next;
    const char *    name;
    const char *    value;
} MCC_SDP_ATTRIBUTE;
```

Fields

Field	Type	Description	SDP field	Required/optional
next	MCC_SDP_ATTRIBUTE	Pointer to the next attribute in the linked list, if available.	a	Optional
name	const char	Attribute name.	a	Required
value	const char	Attribute value.	a	Optional

MCC_SDP_FORMAT

Represents an SDP media format. When an SDP message contains multiple media formats, they are represented by a linked list of MCC_SDP_FORMAT structures.

Definition

```
typedef struct _MCC_SDP_FORMAT
{
    struct _MCC_SDP_FORMAT *    next;
    const char *                id;
    MCC_SDP_RTPMAP *            rtpmap;
} MCC_SDP_FORMAT;
```

Fields

Field	Type	Description	SDP field	Required/optional
next	MCC_SDP_MEDIA	Pointer to the next media format in the linked list, if available.	N/A	Optional
id	const char	Media format payload identifier, as a string constant. For audio and video media, use the media payload type as defined in the RTP Audio/Video profile (RFC 1890).	a	Required

Field	Type	Description	SDP field	Required/optional
rtpmap	MCC_SDP_FORMAT	Pointer to the first rtpmap attribute that corresponds to this format, as defined by the MCC_SDP_RTPMAP structure. The MCC SDP library automatically creates rtpmap attributes for this structure as follows: • If the SDP message has media-level rtpmap attributes, then the library copies the media-level rtpmap attributes to this structure. • If the SDP message has only session-level rtpmap attributes, then the library copies the session-level rtpmap attributes to this structure. The value of the rtpmap field is 0 if there are no rtpmap attributes associated with the SDP message.	a	Optional

MCC_SDP_MEDIA

Represents a media description in an SDP message. If the SDP message contains multiple media descriptions, they are represented by a linked list of MCC_SDP_MEDIA structures.

Definition

```
typedef struct _MCC_SDP_MEDIA
{
    struct _MCC_SDP_MEDIA * next;
    const char * type;
    const char * protocol;
    const char * info;
    uint32_t port;
    uint32_t count;
    MCC_SDP_ADDRESS * connection;
    MCC_SDP_ATTRIBUTE * attr;
    MCC_SDP_RTPMAP * rtpmap;
    MCC_SDP_FORMAT * format;
} MCC_SDP_MEDIA;
```

Fields

Field	Type	Description	SDP field	Required/optional
next	MCC_SDP_MEDIA	Pointer to the next media description in the linked list, if available.	N/A	Optional
type	const char	Media type. Recognized values are: MCCSDP_MEDIA_AUDIO MCCSDP_MEDIA_VIDEO MCCSDP_MEDIA_ATP MCCSDP_MEDIA_DATA MCCSDP_MEDIA_IMAGE MCCSDP_MEDIA_CONTROL	m	Required
protocol	const char	Transport protocol. Recognized values are: MCCSDP_PROT_RTPAVP MCCSDP_PROT_UDP MCCSDP_PROT_TCP MCCSDP_PROT_UDPTL	m	Required
info	const char	Media stream description.	i	Optional
port	uint32_t	Destination port number.	m	Required
count	uint32_t	Port count. The value of this field is 0 if there is no port count.	m	Optional
connection	MCC_SDP_ADDRESS	Media-level connection address.	c	Optional
attr	MCC_SDP_ATTRIBUTE	Pointer to the first media-level general attribute, as defined by the MCC_SDP_ATTRIBUTE structure. The value of this field is 0 if there are no general session-level attributes associated with this SDP message.	a	Optional

Field	Type	Description	SDP field	Required/optional
rtpmap	MCC_SDP_RTPMAP	Pointer to the first media-level rtpmap attribute, as defined by the MCC_SDP_RTPMAP structure. The value of this field is 0 if there are no media-level rtpmap attributes.	a	Optional
format	MCC_SDP_FORMAT	Pointer to the first media format, as defined by the MCC_SDP_FORMAT structure.	m	Required

MCC_SDP_RTPMAP

Represents a session-level or media-level rtpmap attribute. MCC_SDP_RTPMAP represents:

- A session-level rtpmap attribute, when it is used as a parameter in the [MCC_SDP](#) structure.
- A media-level rtpmap attribute, when it is used as a parameter in the [MCC_SDP_MEDIA](#) structure.

When an SDP message contains multiple rtpmap attributes of the same type (session-level or media-level), they are represented by a linked list of MCC_SDP_RTPMAP structures.

Definition

```
typedef struct MCC_SDP_RTPMAP
{
    struct MCC_SDP_RTPMAP * next;
    const char * format;
    const char * encoding;
    uint32_t clockrate;
    const char * parameter;
} MCC_SDP_RTPMAP;
```

Fields

Field	Type	Description	SDP field	Required/optional
next	MCC_SDP_RTPMAP	Pointer to the next rtpmap attribute, if available.	N/A	Optional

Field	Type	Description	SDP field	Required/optional
format	const char	Media format payload identifier, as a string constant. For audio and video media, use the media payload type as defined in the RTP Audio/Video profile (RFC 1890).	a	Required
encoding	const char	RTP encoding name. Recognized values for encoding names include payload names defined in RFC 1890, and RTP payload names registered as MIME types. If the transport protocol is RTP/AVP, then the value of this field is usually a media payload type, as defined in the RTP audio/video profile. For example: PCMU. If the transport protocol is UDP, then the value of this field is usually a MIME type.	a	Required
clockrate	uint32_t	RTP encoding clock rate.	a	Required
parameter	const char	RTP encoding parameter.	a	Optional

11. Using the demonstration program

Interactive test: ctasip

The demonstration program *ctasip* is an interactive program that enables you to perform the following tasks:

- Register SIP users
- Place SIP calls
- Answer incoming SIP calls
- Terminate SIP calls

This program demonstrates how to use the SIP for NCC API. In addition to SIP, *ctasip* supports Fusion. Because of the large number of parameters used in SIP and Fusion, modify one of the configuration files that are included with this product to suit the environment in which *ctasip* is to be run. For a list of these files and their location, refer to [Configuration files](#).

Usage

```
ctasip [options]
```

where **options** are:

Option	Use this option to...
-b boardnumber	Specify the board number assigned in the OAM API configuration file.
-f filename	Specify a NaturalAccess configuration file.
-i filename	Specify a configuration file that contains a sequence of commands to execute.
-s stream[:timeslot]	Specify the stream and optionally the timeslot to use for SIP. If both a stream and timeslot are specified, the format is -s stream:timeslot , for example: <pre>-s 0:5</pre>
-S	Execute the op and sp commands automatically when starting <i>ctasip</i> .
-P filename	Specify the <i>ctasip</i> configuration file to use (recommended). For example, to run <i>ctasip</i> with the <i>ctasip.cfg</i> configuration file, enter the following command: <pre>ctasip -P ctasip.cfg</pre>
-D parameter	Override the specified <i>ctasip</i> configuration parameter. For example: <pre>ctasip -P ctasip.cfg -D mspp.hmp=0</pre>

Option	Use this option to...
-V	Display the <i>ctasip</i> configuration parameters to the console. For example: <pre>ctasip -P ctasip.cfg -V</pre> For more information, refer to ctasip parameters .

ctasip commands

ctasip is a menu-driven interactive program. Enter one- and two-letter commands to execute NaturalAccess, NCC API, ADI API, or VCE API commands. Some commands prompt for additional information, such as frequencies and amplitudes for tone generators. For more information about the service commands, refer to the service-specific reference manuals.

The following table describes the available commands:

Function	Command	Description
Help	h	Displays a table of available commands.
Modify parameters	mp	Modifies <i>ctasip</i> configuration parameters.
Print parameters	pp	Displays <i>ctasip</i> configuration parameters.
Quit	q	Exits from the <i>ctasip</i> program.
Accept call	cc	Calls nccAcceptCall .
Acknowledge PRACK	ap	Calls nccSendPrackResponse .
Answer call	ac	Calls nccAnswerCall .
Call progress begin	cb	Calls adiStartCallProgress .
Call progress stop	cs	Calls adiStopCallProgress .
Call status	c?	Calls nccGetCallStatus .
Collect digits	cd	Calls adiCollectDigits .
Collect stop	sc	Calls adiStopCollection .
Digit flush	df	Calls adiFlushDigitQueue .

Function	Command	Description
Digit get	dg	Calls adiGetDigit .
Digit peek	dp	Calls adiPeekDigit .
Disable DTMF detector	dd	Calls adiStopDTMFDetector .
Disable energy detector	de	Calls adiStopEnergyDetector .
Disable MF detector	dm	Calls adiStopMFDetector .
Disable tone detector	dt	Calls adiStopToneDetector .
Disconnect call	dc	Calls nccDisconnectCall .
Enable DTMF detector	ed	Calls adiStartDTMFDetector .
Enable energy detector	ee	Calls adiStartEnergyDetector .
Enable MF detector	em	Calls adiStartMFDetector .
Enable tone detector	et	Calls adiStartToneDetector .
FSK abort receive	fa	Calls adiStopReceivingFSK .
FSK receive	fr	Calls adiStartReceivingFSK .
FSK send	fs	Calls adiStartSendingFSK .
Generate DTMFs	gd	Calls adiStartDTMF .
Generate net tone	gn	Calls adiStartTones .
Generate stop	gs	Calls adiStopTones .
Generate user tone	gt	Calls adiStartTones .

Function	Command	Description
Get call status	l?	Calls nccGetLineStatus .
Modify play gain	mg	Calls vceSetPlayGain .
Modify play speed	ms	Calls vceSetPlaySpeed .
Open services	op	Calls ctaCreateContext and ctaOpenServices .
Place call	pc	Calls nccPlaceCall .
Place call on hold	oc	Calls nccHoldCall .
Play file	pf	Calls vceOpenFile and vcePlayMessage .
Play memory	pm	Calls vcePlayMessage .
Play/receive status	p?	Calls vceGetContextInfo .
Play/receive stop	ps	Calls vceStop .
Play/record status	r?	Calls vceGetContextInfo .
Play/record stop	rs	Calls vceStop .
Query capability	cq	Calls nccQueryCapability .
Record file	rf	Calls vceCreateFile and vceRecordMessage .
Record memory	rm	Calls vceCreateMemory , vceEraseMessage , and vceRecordMessage .
Reject call	jc	Calls nccRejectCall .
Release call	rc	Calls nccReleaseCall .
Retrieve call	rr	Calls nccRetrievecall .
Send ACK	ak	Calls nccAcknowledgeCall .

Function	Command	Description
Send capability query	sq	Calls nccSendCapabilityQuery
Send capability response	cr	Calls nccSendCapabilityResponse
Send PRACK	pk	Calls nccSendPrack .
Send REGISTER	ru	Calls nccRegisterUser .
Set parameters	pd	Calls ctaGetParmByName and ctaGetParmInfo .
Start protocol	sp	Calls nccStartProtocol or adiStartProtocol .
Stop protocol	up	Calls nccStopProtocol or adiStopProtocol .
Supervised transfer	st	Calls nccTransferCall .
Transfer call	tc	Calls nccAutomaticTransfer .
View parameters	vd	Calls ctaGetParmID and ctaGetParmInfoEx .

ctasip parameters

To display a list of *ctasip* parameters, enter the following command:

```
ctasip -V
```

The following table describes these parameters:

Parameter	Specifies...
general.board	Board number assigned in the OAM API configuration file.
general.slot	Timeslot to use for SIP. This value can range from 0 to the number of SIP licenses and must be unique to each instance of <i>ctasip</i> that is running.
general.autoStart	When the value is a non-zero value, the op (open port) and sp (start protocol) commands run automatically when <i>ctasip</i> starts.
general.autoRelease	When the value is a non-zero value, nccReleaseCall is called automatically after receiving a disconnect event.

Parameter	Specifies...
sip.to	Default To address (called) for nccPlaceCall and also a default address for nccRegisterUser . The default address is 127.0.0.1, which works for placing calls locally to another instance of <i>ctasip</i> . Change this address to the actual IP address of the host machine.
sip.from	Default From address (calling) for nccPlaceCall and also a default address for nccRegisterUser . The default address is 127.0.0.1 (refer to the description of sip.to).
sip.registrar	Default address for a SIP registrar. This parameter is optional. Specify a valid address for a SIP registrar if nccRegisterUser is used.
sip.contact	Default contact address for the nccRegisterUser . Specify a valid contact address if nccRegisterUser is used.
sip.auth.user	Default user name if authentication is used either by a SIP registrar or for outgoing INVITE messages.
sip.auth.password	Default password if authentication is used either by a SIP registrar or for outgoing INVITE messages.
sip.sdp.connection.networkType	Default network type in the connection portion of the outgoing SDP information.
sip.sdp.connection.addressType	Default address type in the connection portion of the outgoing SDP information.
sip.sdp.connection.address	Default connection address used in passing SDP information. If using a CG board and Fusion, change this address to the address assigned to the CG board.
sip.sdp.connection.port	Default connection port for SDP information. If multiple versions of <i>ctasip</i> are used, this address must be different. Note: When specifying this address, two ports are used: the even numbered port carries media, and the odd numbered port carries control information.
sip.sdp.origin.username	Default user name.
sip.sdp.origin.sessionId	Default session ID.
sip.sdp.origin.version	Default version number.

Parameter	Specifies...
sip.sdp.origin.networkType	Default network type. IN means internet.
sip.sdp.origin.addressType	Default address type. Only IPv4 is supported.
sip.sdp.origin.address	Default IP address.
mspp.hmp	Set to zero for Fusion.
mspp.nomedia	<i>ctasip</i> has the capability of running in a pure call control mode. Set this value to a non-zero value if only signaling is being used.
mspp.slot	Timeslot to use to create a context for Fusion. Use this parameter only if you are using Fusion. This value cannot be the same as the general.slot parameter.
voice.play.file	Default voice file to play using the pf command.
voice.play.type	Default file type. Refer to the <i>Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual</i> .
voice.play.encoding	Default encoding format. Refer to the <i>Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual</i> .
voice.record.file	Default voice file to record using the rf command.
voice.record.type	Default file type. Refer to the <i>Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual</i> .
voice.record.encoding	Default encoding format. Refer to the <i>Dialogic® NaturalAccess™ Voice Control Element API Developer's Manual</i> .

Outbound call example

The following example shows typical output generated by *ctasip* when the program places an outbound call.

To run the program, enter the following command:

```
ctasip -P mysip.cfg
```

ctasip displays information similar to the following:

```
=====
SIP Demo Program
=====
Dialogic Corporation, 2009
=====

Type 'h' for help.
op
ok
Event: CTAEVN_OPEN_SERVICES_DONE, Finished
sp
Enter protocol name ['sip0']:
ok
Event: NCCEVN_START_PROTOCOL_DONE, CTA_REASON_FINISHED
ru
Contact ['sip:4801@10.122.20.20']:
ok
Event: NCCEVN_REGISTER_USER
pc
To ['sip:4800@sipserver.dialogic.com']:
==> Add SDP using SIP IEs...
ok
Event: NCCEVN_PLACING_CALL
Event: NCCEVN_CALL_PROCEEDING
Event: NCCEVN_REMOTE_ALERTING
Event: NCCEVN_CALL_CONNECTED
MSPP : Created: PM endpoint, slot 0 board 0
MSPP : Created: RTP endpoint, 10.122.20.20:8004 -> 10.122.20.22:5004
MSPP : Created: channel
MSPP : Channel: connected and enabled
dc
ok
Event: NCCEVN_CALL_DISCONNECTED
MSPP : Channel: disabled and disconnected
MSPP : Destroyed: RTP endpoint
MSPP : Destroyed: PM endpoint
MSPP : Destroyed: channel
rc
ok
Event: NCCEVN_CALL_RELEASED
```

The following table describes the sequence of commands:

Sequence	Command	Description
1	op	Opens a port as specified in the configuration file. The CTAEVN_OPEN_SERVICES_DONE event is returned.
2	sp	Starts the SIP protocol. The NCCEVN_START_PROTOCOL_DONE event is returned.

Sequence	Command	Description
3	ru	Registers this user with the SIP server. The NCCEVN_REGISTER_USER event is returned.
4	pc	Places a call. The following events are returned: NCCEVN_PLACING_CALL: SIP stack sent out the request. NCCEVN_CALL_PROCEEDING: Remote side sent a Trying response. NCCEVN_REMOTE_ALERTING: Remote side sent a 180 Ringing response. NCCEVN_CALL_CONNECTED: Remote side responded with a 200 OK response to the INVITE generated by nccPlaceCall .
5	dc	Disconnects the call. The remote side responds to the BYE message with a 200 OK response that generates the NCCEVN_CALL_DISCONNECTED event.
6	rc	Releases the call context. The SIP stack internally generates the NCCEVN_CALL_RELEASED response to nccReleaseCall .

Inbound call example

The following example shows typical output generated by *ctasip* when the program places an inbound call:

```

Event: NCCEVN_SEIZURE_DETECTED
Event: NCCEVN_INCOMING_CALL
TO : <sip:4801@sipserver.dialogic.com>
FROM: "SIPPhone"<sip:4800@sipserver.dialogic.com>
cc
ok
Event: NCCEVN_ACCEPTING_CALL
ac
ok
Event: NCCEVN_ANSWERING_CALL
Event: NCCEVN_CALL_CONNECTED
MSPP : Created: PM endpoint, slot 0 board 0
MSPP : Created: RTP endpoint, 10.122.20.20:8004 -> 10.122.20.22:5004
MSPP : Created: channel
MSPP : Channel: connected and enabled
Event: NCCEVN_CALL_DISCONNECTED
MSPP : Channel: disabled and disconnected
MSPP : Destroyed: RTP endpoint
MSPP : Destroyed: PM endpoint
MSPP : Destroyed: channel
rc
ok
Event: NCCEVN_CALL_RELEASED

```

The following table describes the sequence of events and commands:

Sequence	Event or command	Description
1	NCCEVN_SEIZURE_DETECTED	Indicates a new call has arrived as the result of an incoming INVITE message.
2	NCCEVN_INCOMING_CALL	Carries the called (To) and calling (From) information along with the SDP information.
3	cc	Sends a 180 Ringing response to the remote SIP phone. The event NCCEVN_ACCEPTING_CALL is generated by the SIP stack indicating that the response was successfully sent.
4	ac	Answers the call by sending a 200 OK response to the initial INVITE.
5	MSPP events	Result from setting up an MSPP endpoint.
6	NCCEVN_CALL_DISCONNECTED	Indicates that the remote end disconnected. The SIP stack automatically acknowledges this call.
7	rc	Calls nccReleaseCall to free up the SIP context.

12. Index

A

accepting a call 53
accepting a transfer request 62
acknowledging a call 57
acknowledging a PRACK 61
answering a call 52
applications 13
 building 15
 exchanging messages 28

C

call status 55
call transfer 58
capability information 63, 64
capability mask 73
client-server model 28
command line 17, 26
components 11
configuration files 12
configuration parameters 19
configuring the server 17
contexts 14
creating an SDP message 76
CTA_EVENT 33
ctasip 101
ctasip commands 102
ctasip parameters 105
ctaWaitEvent 33

D

data structures 34
demonstration program 12, 101
disconnecting a call 55

E

environment 13
error codes 41
event queues 14

events 33, 66
exchanging messages 28
 establishing a session 29
 receiving messages 31
 requests 28
 responses 28
 sending messages 29
 terminating a session 29
extended call status 56

F

Fusion 14

H

header files 11, 35, 41

I

inbound calls 70, 109
information element (IE) 36
 data structures 34
 values 40

K

keywords 19

L

libsipmgr.so 11
license 13
Linux 26
local disconnect 72
logging 27

M

MCC_SDP 75, 79, 93
MCC_SDP_ADDRESS 95
MCC_SDP_ATTRIBUTE 95
MCC_SDP_FORMAT 96
MCC_SDP_MEDIA 97
MCC_SDP_RTPMAP 99
mccdPrintSDP 79, 80
mccdReadSDP 82

mccdSdpAddAttribute	76, 77, 83	nccRegisterUser	62
mccdSdpAddConnIP4	76, 77, 84	nccRejectCall	54
mccdSdpAddConnIP6	76, 77, 84	nccRetrieveCall	59
mccdSdpAddInformation	76, 77, 85	nccSendCapabilityQuery	63
mccdSdpAddMediaBegin	77, 86	nccSendCapabilityResponse	64
mccdSdpAddMediaEnd	77, 87	nccSendPrack	19, 61
mccdSdpAddMediaFormat	77, 87	nccSendPrackResponse	19, 61
mccdSdpAddRtpmap	76, 77, 88	nccStartProtocol	15
mccdSdpAddSessionIP4	76, 89	nccTransferCall	58
mccdSdpAddSessionIP6	76, 89	nccxsip.h	11, 35, 41
mccdSdpGetLength	76, 90	nmssip	11, 28
mccdSdpInit	76, 91	O	
mccReadSDP	75	on hold	59
mccSdpAddSessionIP4	89	outbound calls	71, 108
media streaming	16	P	
messages	28	placing a call	47
formats	35	PRACK	61
N		printing an SDP message	79
Natural Access	13	protocol layers	10
NCC functions	34, 44	R	
NCC libraries	11	reading an SDP message	75
NCC_SIP_INFO	34	readme file	11
NCC_SIP_INFO_ADD_NUM	35	readme_sip.txt	11
NCC_SIP_INFO_ADD_STR	35	receiving SIP messages	31
NCC_SIP_INFO_INIT	35	registering a user	62
NCC_START_PARMS	33	rejecting a call	54
nccAcceptCall	53	rejecting a transfer request	62
nccAcknowledgeCall	19, 57	remote disconnect	73
nccAnswerCall	52	requests	28
nccAutomaticTransfer	58	responses	28, 47
nccDisconnectCall	55	retrieving a call	59
nccdSdpAddConnIP6	89	S	
NCCEVN_	66	SDP	35
nccGetCallStatus	55	creating an SDP message	76
nccGetExtendedCallStatus	56	encoding information	28
nccHandleTransferRequest	62	function reference	80
nccHoldCall	59	MCC SDP library	75
nccPlaceCall	35, 47	printing an SDP message	79

reading an SDP message	75	SIP stack	11
structure hierarchy	92	command line options	17
sending a PRACK.....	61	configuration parameters.....	19
sending SIP messages	29	SIP_IE_METHOD	39
sequence diagrams.....	69	SIP_IE_RESPONSE_CODE	39
service events	66	SIP_IE_SDP_ENCODING	41
session description protocol (SDP).....	28	SIP_IE_SDP_MEDIA_PORT	40
sessions	29	SIP_IE_SDP_MEDIA_PROTOCOL	41
SIP	10	SIP_IE_SDP_MEDIA_TYPE	40
command line options	17	sipmgr.dll.....	11
configuration parameters.....	19	Solaris	26
events	33	T	
protocol layers	10	transfer requests.....	62
receiving messages	31	transferring a call	58
sending messages	29	W	
stack	11	Windows	26