Dialogic®
Making Innovation Thrive™

Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual

October 2010 64-0502-03 Rev A

www.dialogic.com

Table Of Contents

Copyright and legal notices

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
9000-6708-18	May, 2002	MVH, NACD, 2002-1
9000-6708-19	April, 2003	LBG, Natural Access 2003-1
9000-6708-20	December, 2003	LBG, Natural Access 2004-1 beta
9000-6708-21	April, 2004	MCM, Natural Access 2004-1
9000-6708-22	March 2005	LBG, Natural Access 2005-1
9000-6708-23	February 2009	DEH, Natural Access R8.1
64-0502-01	October 2009	LBG, NaturalAccess R9.0
64-0502-02	December 2009	LBG, NaturalAccess R9.0.1
64-0502-03 Rev A	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.

1. Introduction

This manual describes how to use the NaturalAccess™ NaturalCallControl™ API to perform call control within a NaturalAccess telephony application, and provides detailed descriptions of the NCC API capabilities and functions. Use this reference manual with the *Dialogic® NaturalAccess™ Software Developer's Manual*.

This manual is targeted to developers of telephony and voice applications who are using NaturalAccess. This document defines telephony terms where applicable, but assumes that you are familiar with telephony concepts and switching. It also assumes that you are familiar with 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. Overview of NaturalCallControl

NCC API features

Call control, an essential part of telephony applications, provides a telephony network connection between two endpoints so that a telephone call can proceed. NCC is a NaturalAccess API for writing applications that perform call control.

Call control features include:

- Detecting, and answering or rejecting an incoming (inbound) call
- Placing an outgoing (outbound) call
- Transferring one call to another
- Placing a call on hold, retrieving a call from hold
- Disconnecting a call
- Blocking all calls on a trunk channel
- Querying the status of a trunk channel or call

Call control (and therefore the NCC API) is not concerned with what happens once the call is connected (for example, voice file playing/recording, sending or receiving faxes). In the NaturalAccess model, telephony switching is outside the realm of call control, although switching may occur during call control.

Call control can be conducted in a variety of ways depending on the protocol that is running on the system. NaturalAccess implements specific protocols through trunk control programs that run on AG Series and CG Series board DSPs.

Protocols

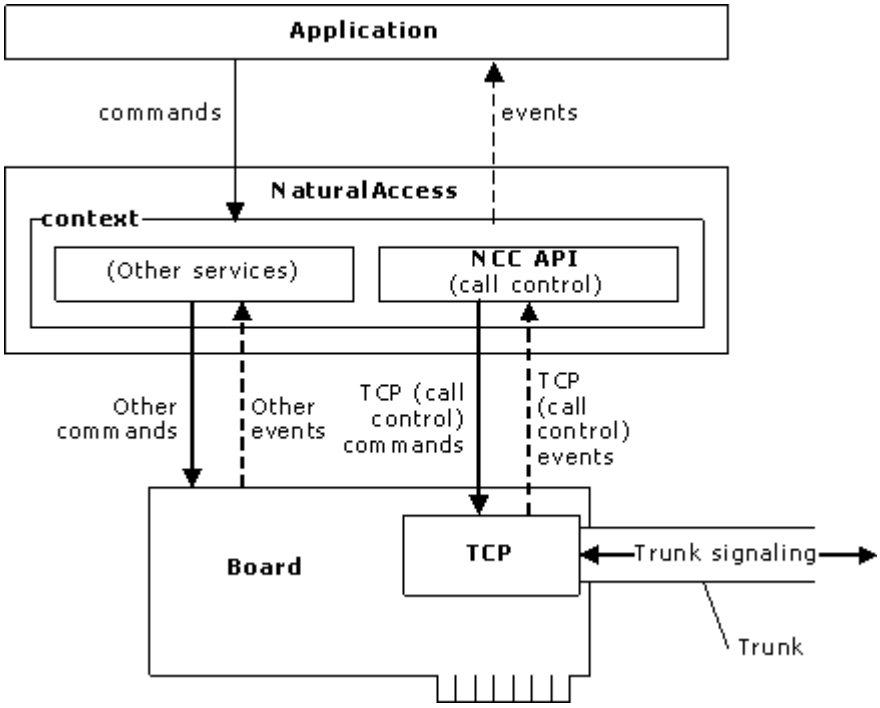
Call control is implemented differently in different situations, depending upon characteristics such as the network type and country. Each specific implementation of call control is called a protocol. Protocols are typically subdivided into groups based on signaling method (CAS, ISDN, SS7). Within each group, there are sub-groups, or variants such as loop start, MFC-R2, and 4ESS. The NCC API is designed to be largely protocol-independent: applications use the same NCC functions regardless of the protocol. However, an application can send protocol-dependent commands and information using the NCC API. For more information, see Protocol capabilities.

Trunk control programs

Most NaturalAccess APIs interact with telephony hardware in some fashion. For example, the Switching API controls the H.100 or MVIP switching chip on a telephony board. The Voice Message API uses a board's DSP resources to play and record files. The NCC API interacts with a board's processing resources to perform call control on trunks connected to the board either directly through the board's ports or indirectly through the H.100 or MVIP bus.

The method used by the NCC API to interact with hardware differs depending upon the implementation. For example, with the ADI implementation, a protocol-specific trunk control program (TCP) is downloaded to the board at startup. The TCP interacts directly with the trunk, interpreting commands from the NCC API in a protocol-specific manner, and translating protocol-specific trunk events and errors into standard NCC API events.

The following illustration shows the NCC API interacting with hardware:



One or more TCPs is provided for each protocol. You load the TCP into the on-board memory of a line interface board. For applications that must simultaneously support multiple protocols and/or protocol variations, more than one TCP can be loaded to the telephony board at the same time. Each line supports one TCP at a time.

The TCPs to be loaded for an application are specified in the configuration file. When you run your board configuration utility, it downloads the specified TCPs to the board. For more information about configuring your boards, see your board-specific installation and developer's manual.

Some protocols do not support call transfer; other protocols do not support call hold/retrieve. Once a TCP is loaded to the board, the application can determine the capabilities of the TCP using nccQueryCapability. For more information, see Protocol capabilities.

NCC API call control model

The NCC API call control model differentiates between lines and calls:

- A line is a logical representation of a channel on a trunk.
- A call is a connection between two parties, a connection in the making, or a former connection on a line. Multiple calls can exist simultaneously on a line. However, only one call at a time can be active (not disconnected or held).
- A line is referenced using a line handle. The line handle is equivalent to the context handle.
- A call is referenced using a call handle. When a call handle is referenced in a function call, the line handle is referenced implicitly.

When a line event occurs, the event indication includes the line handle. When a call event occurs, the event indication includes both the line handle and call handle. For more information about receiving NCC API events, see Receiving NCC API events.

In the NCC API call control model, the state is the condition or status of a line or call. The model defines a set of specific states that a line or call can be in as long as it exists. For each state, a specific function call by the application, or actions by the remote party is defined, that can cause the line or call to change to another state. Whenever a state change occurs, the application receives an appropriate event.

To determine the current state of a call or line, the application can invoke status retrieval functions. For more information, see Getting status information.

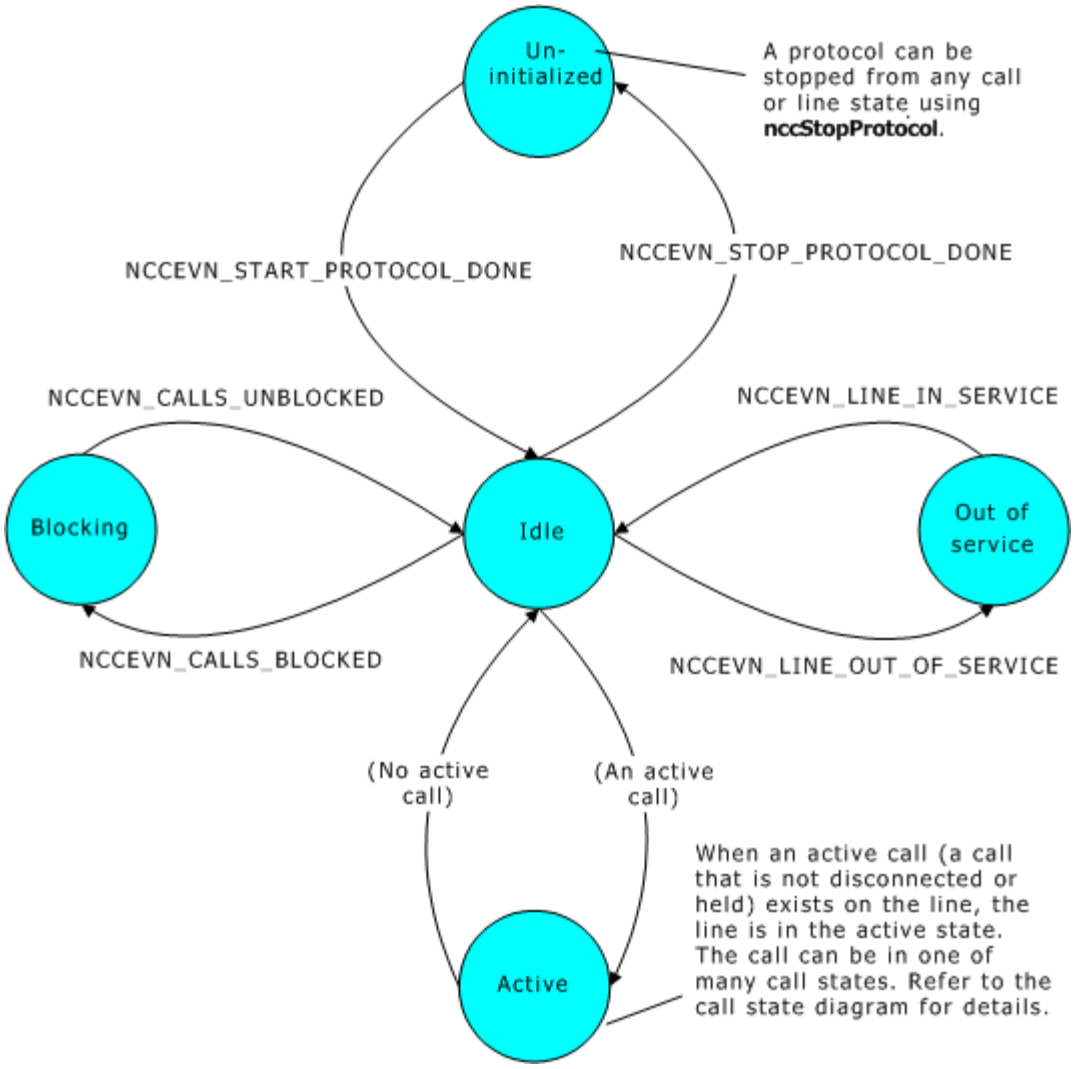
Line states

The following table lists the line states. One or more transitional events associated with each state indicate transition into the state. An application can use nccGetLineStatus to determine the state of a line:

A line enters this state...	In this way...
Active	• NCCEVN_SEIZURE_DETECTED event, when a call is coming in. • By attempting to place a call using nccPlaceCall. • NCCEVN_CALL_RETRIEVED event, which means a call is active and the line has reentered active state.
Blocking	NCCEVN_CALLS_BLOCKED event, which is solicited by nccBlockCalls.
Idle	• NCCEVN_START_PROTOCOL_DONE event, which is solicited by nccStartProtocol. • NCCEVN_CALLS_UNBLOCKED event, which is solicited by nccUnblockCalls. • NCCEVN_LINE_IN_SERVICE event (an unsolicited event). • NCCEVN_CALL_HELD or NCCEVN_CALL_DISCONNECTED event, from the active state. These events mean that a call was placed on hold or is disconnected, and is not active. When no calls are active on a line, the line enters idle line state.
Out of service	NCCEVN_LINE_OUT_OF_SERVICE event, which is an unsolicited event.
Uninitialized	• Initial state of line. When the NCC API is opened on a context, the line handle (signified by the handle) is created in an uninitialized state. • NCCEVN_STOP_PROTOCOL_DONE event, which is solicited by nccStopProtocol.

If all calls on a line are held or in disconnected call state so that all calls are inactive, the line state changes to idle. If a held call is retrieved or a new call comes in or is placed, the line state returns to active.

The following illustration shows the line states, and events indicating transitions between them:



Call states

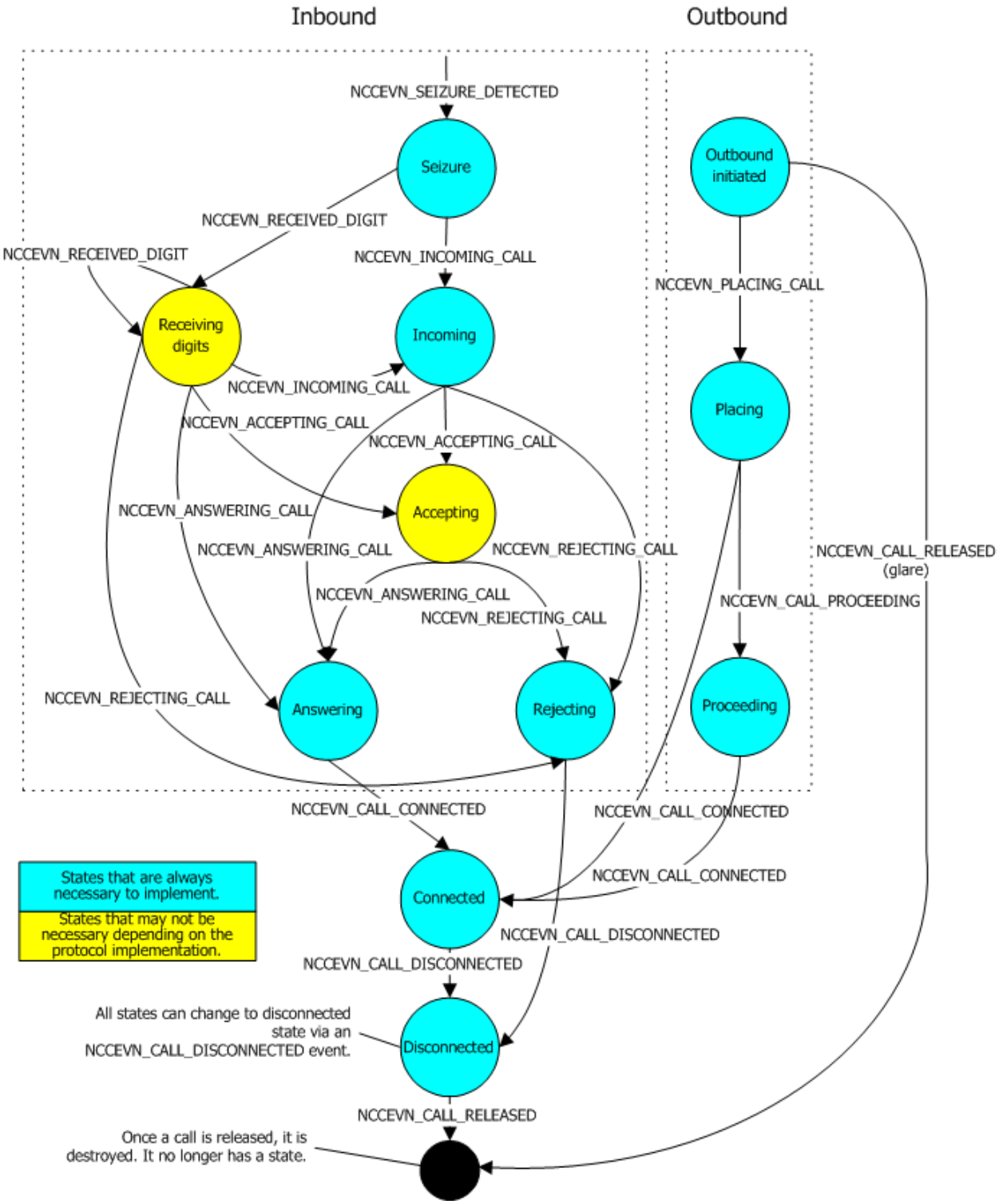
The following table lists the call states. One or more transitional events associated with each state indicate transition into the state.

Some call states are optional. If a call enters an optional state, the event indicating the transition is generated only if the proper bit enabling or disabling the event is set in the NCC.START.eventmask parameter. For more information about this parameter, see NCC API global parameters.

The application can use nccGetCallStatus to determine the state of a call.

A call enters this state...	In this way...
Accepting (optional)	NCCEVN_ACCEPTING_CALL event in response to invocation of nccAcceptCall. The NCC_CAP_ACCEPT_CALL indicator in the capabilitymask returned by nccQueryCapability indicates if the protocol supports this state.
Answering	NCCEVN_ANSWERING_CALL event in response to invocation of nccAnswerCall.
Connected	Unsolicited NCCEVN_CALL_CONNECTED event after a call is successfully answered by the remote party, or the NCCEVN_CALL_CONNECTED call control connectmask conditions occurred on an outbound call (connect on proceeding).
Disconnected	• Unsolicited NCCEVN_CALL_DISCONNECTED event from any state. • Solicited NCCEVN_CALL_DISCONNECTED event, following the invocation of nccDisconnectCall. The NCC_CAP_DISCONNECT_IN_ ANY_STATE indicator in the capabilitymask returned by nccQueryCapability indicates in which call states the application can initiate a disconnect.
Incoming	NCCEVN_INCOMING_CALL event.
Outbound initiated	By attempting to place a call using nccPlaceCall.
Placing	NCCEVN_PLACING_CALL event in response to invocation of **nccPlaceCall**.
Proceeding	An NCCEVN_CALL_PROCEEDING event indicates that the call entered this state.
Receiving digits (optional)	Unsolicited NCCEVN_RECEIVED_DIGIT event. This event is generated only if the NCC.START.overlappedreceiving parameter is set. For more information, see NCC API global parameters.
Rejecting	• NCCEVN_REJECTING_CALL event in response to invocation of nccRejectCall. • As a result of not responding in time to an NCCEVN_INCOMING_CALL.
Seizure	Unsolicited NCCEVN_SEIZURE_DETECTED event.

The following illustration shows the NCC call states and events indicating transitions between them:



3. Initializing an NCC API application

Setting up the NaturalAccess environment

Before you can call functions from the NCC library, follow these steps to set up the NaturalAccess environment:

1. Initialize NaturalAccess
2. Create event queues
3. Create contexts and attach them to event queues.
4. Open services, including NCC, on each context.

To set up a second NaturalAccess application that shares a context with the first application:

1. Initialize the NaturalAccess application.
2. Create event queues.
3. Attach the application to the existing context.

Once the application has performed these tasks, it can receive events associated with the event queues it has created.

Initializing NaturalAccess

Use **ctaInitialize** to initialize NaturalAccess, registering the services available to the application. Specify the service and service manager names in the call to **ctaInitialize**. The application can open only the services initialized with **ctaInitialize**. Service managers are dynamic link libraries (DLLs) in Windows and shared libraries in UNIX.

When using an AG or a CG board, specify the ADIMGR service manager.

Creating event queues and contexts

After initializing NaturalAccess, create one or more event queues by invoking **ctaCreateQueue**. Specify the service managers to attach to each queue. By attaching a service manager to a queue, you make that service manager available to the queue. After the services are opened, the events generated by the managed service go to the attached queue. **ctaCreateQueue** returns an event queue handle (***ctaqueuehd***) that the application uses to address the event queue.

The next step is to create a context and attach it to an event queue. Invoke **ctaCreateContext** and provide the queue handle (***ctaqueuehd***) returned from **ctaCreateQueue**. **ctaCreateContext** returns a context handle (***ctahd***) that the application uses when invoking NCC service line functions. Events communicated back to the application are also associated with the context.

Refer to the *Dialogic® NaturalAccess™ Software Developer's Manual* for details on the programming models created by the use of contexts and event queues.

Opening services

Invoke **ctaOpenServices** to open services on a context. When opening a service on a context, use the substructures of the CTA_SERVICE_DESC structure to specify information about the service and the service manager, and the resources to attach to the context.

The following table shows the information to enter to start NCC on a context, assuming an implementation using an AG board:

Substructure	Parameter	Set as follows for the ADIMGR implementation of NCC
CTA_SERVICE_NAME	svcname	ncc
	svcmgrname	The name of the implementation DLL for the protocols to be used: ADIMGR.
CTA_SERVICE_ADDR	N/A	Reserved for client/server usage. Initialize to 0.
CTA_SERVICE_ARGS	N/A	Not used by NCC.
CTA_MVIP_ADDR	board	The board number to use. The configuration file contains a Board directive identifying each board in the system. Refer to your board-specific installation and developer's manual for more detail.
	bus	MVIP95_LOCAL_BUS
	stream	See Streams and timeslots.
	timeslot	See Streams and timeslots.
	mode	The value of mode can be: ADI_VOICE_INPUT Receives inband data only. The data is received by the DSP on the given timeslot. ADI_VOICE_OUTPUT Transmits inband data only. The data is transmitted by the DSP on the given timeslot. ADI_VOICE_DUPLEX Receives and transmits inband data on the given timeslot. Typically used with the NOCC protocol and allows media (for example, voice, fax) reception and transmission. ADI_FULL_DUPLEX This mode is both ADI_SIGNAL_DUPLEX and ADI_VOICE_DUPLEX. The port receives and transmits both inband media and out-of-band signaling. Typically used when running a network protocol on the port. Allows both voice (inband) and network signaling transmission and reception.

Streams and timeslots

When you open NCC with ADIMGR, specify a DSP address. A DSP address is specified as a complete MVIP address including a stream/timeslot pair.

The following table shows valid addresses for boards that support NCC:

Board	Stream Bus=MVIP95_LOCAL_BUS	Timeslot
AG 2000	4	0 - 7
AG 2000C	4	0 - 23
AG 2000-BRI	4	0 - 7
CG 6060, CG 6060C	64	0 - 899*
CG 6565, CG 6565C, CG 6565E	64	0 - 899*

*The timeslot's upper limit is configurable. For information, see the board manuals.

The demonstration program *ctatest* verifies the proper installation and operation of the NCC API. Refer to the *Dialogic® NaturalAccess™ Software Developer's Manual* for more information about *ctatest*.

Receiving NCC API events

When the NCC API manager is attached to an event queue, it opens the board driver and associates the muxable wait object returned by the driver open command with the event queue. When this wait object is signaled on receipt of events from the board, **ctaWaitEvent** processes the events through NCC and passes all generated events back to the calling function.

Events arrive in the form of the standard event data structure defined in *ctadef.h*. For more information, see Overview of the NCC API events.

Loading and starting a protocol

After you initialize NaturalAccess and create event queues and contexts, the line represented by each context is in an uninitialized line state. The application now starts a protocol (TCP) instance on each context. This changes the line state to idle, preparing it to accept an incoming call or place an outbound call.

Complete the following steps to start a TCP instance:

Step	Action
1	The NaturalAccess Parameter Management Service loads protocol parameters and default values. For more information, see Overview of the NCC API parameters.
2	The application modifies the values, if necessary.
3	The application calls nccStartProtocol to start the protocol instance. The instance is parameterized according to the loaded parameters.

The following sections provide general information about loading and starting protocols. For specific information about a protocol, refer to the protocol-specific documentation.

Starting a protocol on a context

After a context is open and the protocol parameters are loaded, the application starts a protocol on that context using the loaded parameters. **nccStartProtocol** starts a protocol on a context. If the protocol starts successfully, the application can then use call control functions to place and answer calls on that line.

Note: To start a protocol from within an application, the protocol must have been downloaded to the board at system initialization time. The configuration utility downloads all protocols specified in the configuration file. For more information about the configuration file, see your board-specific installation and developer's manual.

protname passed to **nccStartProtocol** should be the name of the protocol, as specified in the configuration file. For example, for ISDN Software, you would specify isd0. Unless you want to load custom parameters, set ***startparms***, ***mgrstartparms***, and ***protostartparms*** to NULL so the default protocol parameters are used. For more information, refer to NCC API global parameters.

When **nccStartProtocol** is called, NCCEVN_START_PROTOCOL_DONE is returned. If the protocol is started successfully, the event value field contains CTA_REASON_FINISHED. The line state changes from uninitialized to idle. The application can now use other NCC call control functions to perform call control on the line.

Linking with the NCC API

The NCC API contains two components, the NCC API interface and the NCC API implementation. When building a NaturalAccess application that uses the NCC API, link to *nccapi.lib* under Windows and to *libnccapi.so* under UNIX.

For existing applications, modify your makefiles to link with *nccapi.lib* under Windows or *libnccapi.so* under UNIX.

Limiting call attempts by automatic dialers

International regulatory approvals require you to limit attempts by an automated dialer to call a given number. Limits are placed on both the number and the frequency of the call attempts. For example, most European countries restrict the number of call attempts to a given address to 15, with a minimum interval of 5 seconds between each call attempt.

The NCC API provides functions that enable you to comply with these international regulatory requirements. These functions can be accessed by any application using the ADI implementation of NCC.

During installation, repeat dial attempt functionality is disabled by default. Follow this procedure to enable this functionality:

Step	Action
1	Rename *rda.dlx* (located in the *\bin* directory of the installation) to *rda.dll.* If you are running under UNIX, rename *librda.sx* to *librda.so*.
2	Create an *rda.cfg* file specifying the rules for repeat dialing, and place this file in the CTA_DPATH.

Creating rda.cfg

rda.cfg consists of one line with three positive integers, each separated by a space. The first integer specifies the maximum number of calls that can be made in a series to a given address. The second integer specifies the minimum interval, in seconds, between calls to the same address. The third integer specifies the minimum interval, in minutes, between series' of calls to the same address.

For example, an *rda.cfg* with the following line:

```
3 10 1
```

permits a maximum of 3 calls in each series. Each of these calls must be separated by at least 10 seconds. If an application attempts to place a call to an address within 10 seconds of the most recent call, NCCERR_ADDRESS_BLOCKED is returned. After 3 successful calls, 1 minute must pass before a call to the same address is allowed. If the value of the third parameter is 0, no further calls can be placed to the given address until the log is manually reset.

Note: The call attempt series parameters apply to wrong numbers and unsuccessful calls. The maximum interval between call attempts is 12 minutes.

Repeat dial regulations

The following table lists repeat dial regulations by country:

Country	Maximum number of call attempts	Minimum interval (seconds) between call attempts	Minimum interval (minutes) to wait for a new series
European countries	15	5	-
Australia	31	-	30
Bulgaria	12	2	60
Czech Republic	12	5	-
Israel	15	30	-
Japan	3 in 3 minutes, or 15	- 5	3 -
Korea	3	30	-
Malaysia	2	120	-
New Zealand	5	60	-
Singapore	10	60	-
Slovakia	12	60	-
South Africa	-	60	-
Taiwan	2	60	-

In Australia, for equipment with a service tone detector, a maximum of 10 calls is allowed. In New Zealand, not more than five call attempts to the same number within a 60-minute period, with a minimum of 60 seconds between attempts, is allowed. A total of 10 call attempts to the same number is allowed.

4. Processing inbound calls

Inbound call procedures

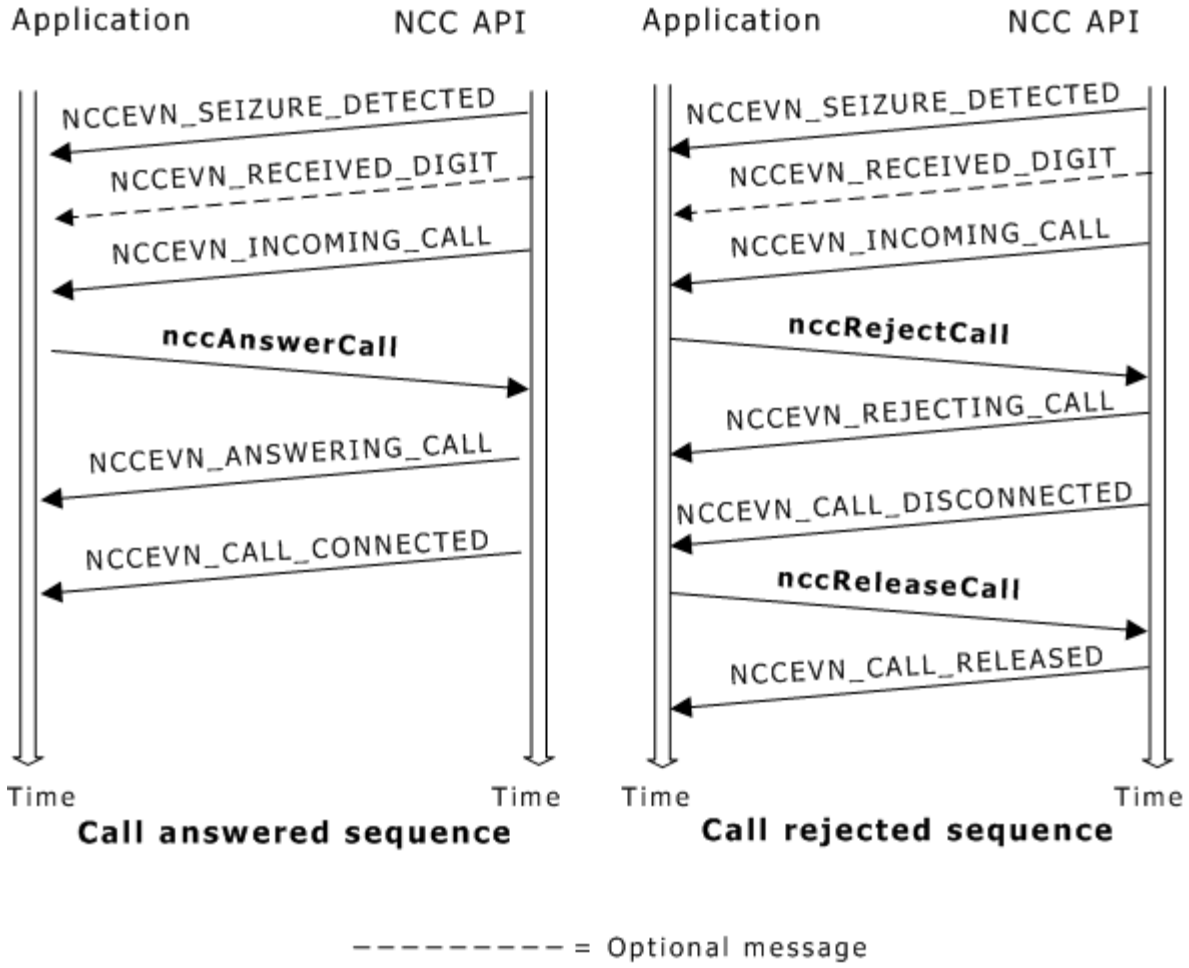
After you start a protocol on a line context with **nccStartProtocol**, the protocol is eligible to receive incoming calls. An inbound call is established in the following way:

Stage	Description
1	The telephone network offers the call, and the call state changes to the seizure call state. NCC generates an NCCEVN_SEIZURE_DETECTED event indicating transition to the seizure call state, and assigns a call handle to the call, to be used for all future activity with this call. The application can retrieve the associated line handle (***ctahd***) by invoking **nccGetCallStatus**. If this is the first call on the line, the line state changes from idle to active.
2	The network delivers the call to the NCC API, and the call state changes to incoming. NCC generates an NCCEVN_INCOMING_CALL event. Depending upon the setting of the NCC.START.overlappedreceiving parameter, the transition to the incoming call state may be preceded by an interval in which the network sends one or more digits (for example, caller ID). For more information, see Receiving overlapped digits.
3	The application decides whether to answer or reject the call, using the **nccAnswerCall** or **nccRejectCall** functions. With some protocols, the application can accept a call immediately without answering or rejecting it, using the **nccAcceptCall** function. This action allows the application to perform media functions (such as playing a voice file) before connecting (or rejecting) the call. To help determine what to do with a call, the application can invoke **nccGetCallStatus** to retrieve the incoming address (caller ID) or other information. For more information, see Getting caller address information (caller ID).
4	The NCC API performs network procedures to execute the application's decision, and returns appropriate events: • If the application does not respond to the NCCEVN_INCOMING_CALL event, the call is rejected. • If the application accepts the call, the NCC API generates NCCEVN_ACCEPTING_CALL. The event value field contains the acceptance method. The call state changes to accepting. From this state, the application can answer or reject the call. (This state may not be valid, depending upon the protocol.) • If the application answers the call, the NCC API generates NCCEVN_ANSWERING_CALL. The call state changes to answering. • If the application rejects the call, the NCC API generates NCCEVN_REJECTING_CALL. The call state changes to rejecting.

Stage	Description
5	If the call is successfully established, the NCC API generates NCCEVN_CALL_CONNECTED. The call state changes to connected.
6	If the connection is unsuccessful, or the application rejects the call, the NCC API generates NCCEVN_CALL_DISCONNECTED. The call state changes to disconnected. The application can then release the call with **nccReleaseCall**.

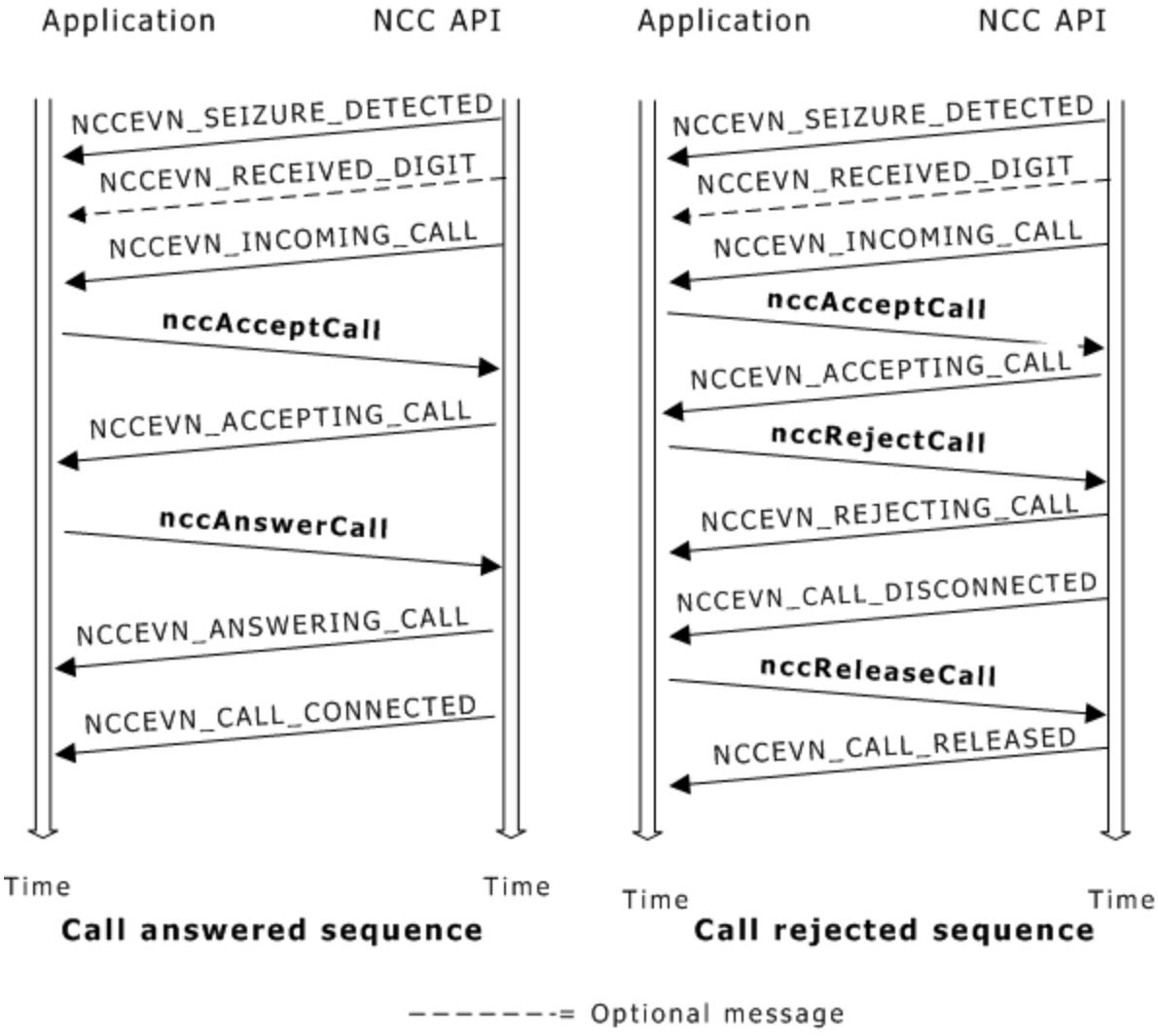
Inbound call sequence

The following illustration shows two protocol timing sequences: one for answering an inbound call, and one for rejecting an inbound call. The illustration shows the normal exchange of commands and events between the NCC API and the application. Optional events are shown with a dashed line:



Inbound call sequence (including nccAcceptCall)

The following illustration shows the same sequences, except the application accepts the call before answering or rejecting it:



Getting caller address information (caller ID)

Most protocols support caller ID: the ability of the called party to determine the address (and sometimes other information) of the calling party. Often, the caller ID service is available from the telephone company on a subscription basis only.

The application can call **nccQueryCapability** to determine if a protocol supports caller ID.

The NCCEVN_INCOMING_CALL event indicates that the protocol has received all calling party information necessary to set up the call. To retrieve caller ID (and other information), invoke **nccGetCallStatus** with a call handle. Information is returned in the NCC_CALL_STATUS structure. Refer to **nccGetCallStatus** for more detail on this structure.

Because the data is stored in this structure on a call-by-call basis, applications must analyze the call status of every incoming call to retrieve call information. Protocol-specific parameters determine how many digits to receive and how to interpret incoming digits. For details, see your protocol documentation.

Receiving overlapped digits

Some protocols support overlapped receiving of digits: the application receives an event as each digit comes in. The application can examine each digit to determine what to do with the call earlier in the call setup process.

The NCC.START.overlappedreceiving parameter in the NCC_START_PARMS structure determines whether the protocol accepts digits in overlapped receiving mode or not. This parameter is referenced within the initial invocation of **nccStartProtocol.**

If the protocol accepts digits in this mode, the application can receive NCCEVN_RECEIVED_DIGIT while the protocol is in the seizure state. The event value field contains a digit received from the network. At this point, the protocol enters the receiving digits call state. It remains in this state until all digits are received. As each digit is received, a new NCCEVN_RECEIVED_DIGIT is generated with the digit. Receipt of NCCEVN_INCOMING_CALL indicates that all digits were received, and the protocol has entered the incoming call state. The complete set of digits is then available in the NCC_CALL_STATUS structure.

The application can use **nccAcceptCall**, **nccAnswerCall**, or **nccRejectCall** to accept, answer, or reject the call at any time while receiving digits.

Accepting calls

Receipt of NCCEVN_INCOMING_CALL indicates that NCC detected the incoming call and gathered all necessary call information. The call is in the incoming call state.

At this point, the application can accept, answer, or reject the call with **nccAcceptCall**, **nccAnswerCall**, or **nccRejectCall**. It can also choose to ignore the call completely. If the call is ignored for a certain interval it is automatically rejected. The NCC.START.waitforPCtime parameter determines this interval.

By invoking **nccAcceptCall**, the call enters the accepting call state, where it has been accepted, but has been neither answered nor rejected. This state allows applications to perform media functions, such as playing a voice file, before connecting the call.

Note: Some protocols do not support **nccAcceptCall** and the accepting call state. The application can call **nccQueryCapability** to determine whether the protocol supports this capability.

With most protocols, arguments passed to **nccAcceptCall** can prompt playing of a ring tone or user audio when the call enters the state. The application can call **nccQueryCapability** to determine if the protocol supports these arguments. The acceptance method to use is specified in the ***method*** argument in the function call. The valid values are:

Method	Action
NCC_ACCEPT_PLAY_RING	Play ring tone.
NCC_ACCEPT_PLAY_SILENT	Play nothing.
NCC_ACCEPT_USER_AUDIO	Allow the application to generate tones and perform voice playback functions.

When the call enters the accepting state, the application receives NCCEVN_ACCEPTING_CALL. The event value field contains the acceptance method. The call remains in the accepting state until the application invokes **nccAnswerCall** or **nccRejectCall**. Receipt of NCCEVN_ANSWERING_CALL or NCCEVN_REJECTING_CALL indicates that the call was answered or rejected, or that the telephone network timed out. The call is no longer in the accepting call state.

The application can receive NCCEVN_CALL_DISCONNECTED while the call is in the accepting call state. This event indicates that the remote party hung up. The event value field contains the reason. The application then invokes **nccReleaseCall** to release the call.

If the remote party disconnects while the application is accepting the call, the application receives NCCEVN_CALL_DISCONNECTED. In this case, NCCEVN_ACCEPTING_CALL is not delivered.

Answering calls

To answer a call, an application invokes **nccAnswerCall**. You can specify a number of ring tones for **nccAnswerCall** to play before answering the call.

Note: Any ring tones generated prior to calling **nccAnswerCall** do not count towards the number of ring tones specified with **nccAnswerCall**.

When **nccAnswerCall** is invoked, the application receives NCCEVN_ANSWERING_CALL. The call is now in the answering call state. The call remains in this state while the NCC API attempts to answer the call and establish a connection. Subsequent receipt of NCCEVN_CALL_CONNECTED means that the call connection attempt was successful, and the call is now in the connected state. NCCEVN_CALL_DISCONNECTED indicates that the call answering attempt was unsuccessful, and the call is in the disconnected state.

If the remote party disconnects while the application is answering the call, the application receives NCCEVN_CALL_DISCONNECTED. In this case, NCCEVN_ANSWERING_CALL is not delivered.

Rejecting calls

To reject a call, an application invokes **nccRejectCall**. With most protocols, the application can also specify a method to be used to reject the call. The following list describes the valid methods:

Method	Description
NCC_REJECT_PLAY_RINGTONE	Play a ring tone until the remote party disconnects.
NCC_REJECT_PLAY_BUSY	Send a backward tone or signal to indicate that the call is busy.
NCC_REJECT_PLAY_REORDER	Send a reorder tone. Reorder usually means that the number is incorrectly formatted, or is not allocated.
NCC_REJECT_USER_AUDIO	The application generates a recorded message, or a special information tone. If the remote party hangs up, the protocol interrupts the application tone or voice file.

When **nccRejectCall** is invoked, the application receives NCCEVN_REJECTING_CALL. The call is now in the rejecting call state. The call remains in this state while the NCC API attempts to reject the call. Subsequent receipt of NCCEVN_CALL_DISCONNECTED indicates that the call rejection was successful. The application now invokes **nccReleaseCall** to release the call.

If the remote party disconnects while the application rejects the call, the application receives an NCCEVN_CALL_DISCONNECTED event. In this case, NCCEVN_REJECTING_CALL is not delivered.

Sending billing information

In any incoming call state, the application can invoke **nccSetBilling** to set the billing information for the incoming call. NCCEVN_BILLING_SET indicates that the billing information was successfully sent to the network. The value field contains the billing rate setting.

Billing is not supported by all protocols. NCC_CAP_SET_BILLING in the capabilitymask returned by nccQueryCapability indicates if the current protocol supports the ability to set the billing for a call.

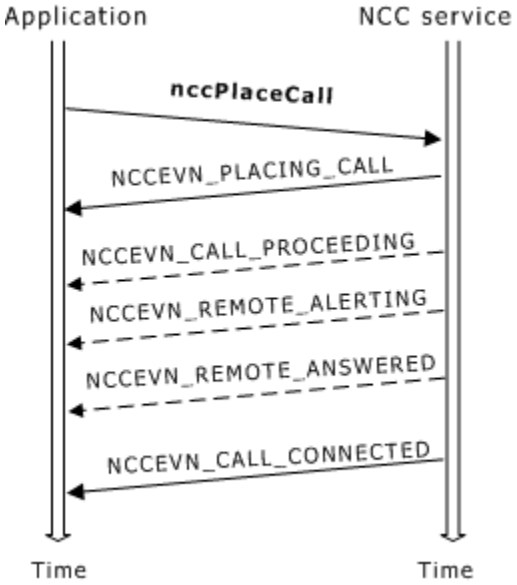
5. Processing outbound calls

Outbound call procedures

After a protocol is started on a line context with **nccStartProtocol**, the application can place outbound calls. An outbound call is established in the following way:

Stage	Description
1	The application invokes **nccPlaceCall**, and the call enters the outbound initiated call state.
2	The protocol checks whether a call collision (glare) situation exists for this call. For example, on two-way trunks, this occurs when the outgoing call collides with an incoming call on the same line. If glare exists, the call is released and NCCEVN_CALL_RELEASED is returned to the application. Otherwise:
3	The protocol seizes the line, creates a call handle for the call, and generates NCCEVN_PLACING_CALL. The line changes to the active line state. The call enters the placing call state
4	NCC initiates dialing using the string specified with **nccPlaceCall**. If the protocol supports overlapped sending of digits, the application can send additional digits and characters using **nccSendDigits**. The NCC_CAP_OVERLAPPED_SENDING bit in the capabilitymask returned by **nccQueryCapability** indicates if the protocol supports this feature.
5	When the switch has accepted the call setup request and is in the process of attempting to ring the receiving end, NCCEVN_CALL_PROCEEDING is returned to the application. The call is now in the proceeding call state. Call progress analysis has begun.
6	When the protocol senses that the remote party is alerting (for example, ringing), NCCEVN_REMOTE_ALERTING is returned to the application. When the protocol senses that the remote party is answering the call, NCCEVN_REMOTE_ANSWERED is returned to the application. **Note:** NCCEVN_REMOTE_ALERTING is generated only if the NCC_REPORT_ALERTING bit is set in the NCC.START.eventmask parameter. NCCEVN_REMOTE_ANSWERED is generated only if the NCC_REPORT_ANSWERED bit is set. For more information, see NCC API global parameters.
7	If a network connection is established, NCC generates NCCEVN_CALL_CONNECTED or NCCEVN_CALL_DISCONNECTED depending upon protocol-specific connectmask and disconnectmask parameters.

The following illustration shows the sequence of command and event interchange for placing an outbound call. Optional events are shown with a dashed line:



Note: International regulatory approvals require you to limit attempts by an automated dialer to call a given number. Limits are placed on both the number and the frequency of the call attempts. For more information, refer to Limiting call attempts by automatic dialers.

Placing an outbound call

To place an outbound call, the application invokes **nccPlaceCall**. The following information is included with this function:

- The address of the remote end of the call to be placed.
- The address of the local end of the call being placed. This can be NULL if the protocol allows.
- Protocol-specific parameters.
- The call handle to assign to the call if it is successfully initialized.

Depending on the protocol, the addresses can be phone number digit strings, IP addresses, or endpoint names. This information can be sent all at once, or, if the protocol supports overlap sending, part of the information can be sent with **nccPlaceCall**, and more can be sent later using **nccSendDigits**, when the call reaches the placing call state. For more information, refer to Sending overlapped digits.

Invocation of **nccPlaceCall** changes the call state to outbound initiated. The call remains in the state until NCCEVN_PLACING_CALL occurs, indicating that the line is seized and glare is resolved.

If the application receives an NCCEVN_CALL_RELEASED event with an NCC_RELEASED_GLARE reason code instead of NCCEVN_PLACING_CALL, the line was seized for an incoming call before glare was resolved and the protocol released the outgoing call. The application immediately abandons outbound call placement and handles the incoming call when the NCCEVN_SEIZURE_DETECTED event for the new call arrives.

Setting call control mask parameters

The call control connectmask and disconnectmask parameters in the NCC_X_ADI_PLACECALL_PARMS structure provide the criteria according to which the NCC API decides to connect or disconnect a successfully placed call. These criteria are protocol-specific. The NCC_X_ADI_PLACECALL_PARMS structure is passed to the NCC API when the call is placed with **nccPlaceCall**.

Different implementations of NCC can provide this functionality in other ways. This section describes how this functionality is provided in the ADIMGR NCC implementation.

For more information about the connectmask or disconnectmask for the protocol you are using, see your protocol-specific documentation.

Connectmask

The NCC_X_ADI_PLACECALL_PARMS connectmask dictates which call progress analysis event causes the NCC API to change the call state from proceeding to connected.

Each of the bits in the connectmask parameter represents a particular condition. If the bit is enabled (set) and the condition described occurs, the call is connected. When the call is connected, the NCC API generates NCCEVN_CALL_CONNECTED. The event's value field indicates the reason for the transition to the connected call state.

Disconnectmask

The NCC_X_ADI_PLACECALL_PARMS disconnectmask parameter dictates which call progress analysis events cause the NCC API to change the call state from proceeding to disconnected.

Each of the bits in the disconnectmask parameter represents a particular condition. If the bit is enabled (set) and the condition described occurs, the NCCEVN_CALL_DISCONNECTED event is generated.

The connectmask takes precedence. If a condition is selected in both the connectmask and disconnectmask, a call is connected if that condition occurs.

Sending overlapped digits

Some signaling protocols such as ISDN support overlapped sending of digits: the sending of address information in multiple bursts, rather than all at once. The application can call **nccQueryCapability** to determine if the protocol supports this feature.

The application supplies the first digit or digits when invoking **nccPlaceCall**. When the application enters the placing call state (see Monitoring the placing call state), it invokes the **nccSendDigits** function repeatedly to send additional segments of the digit string, in order. If the digit string sent by an invocation of either function is not the final one, the final character in the string should be a ~, signifying to the protocol that more digits will follow. The final digit string should not include the final ~ character. For example:

```
nccPlaceCall      172~
nccSendDigits     57924~
nccSendDigits     5~
nccSendDigits     28
```

Final digit string: 17257924528

Note: The overlapped sending feature is not available for the LPS, OPS, and GDS protocols.

Monitoring the placing call state

If the application receives NCCEVN_PLACING_CALL while the call is in the outbound initiated call state, the protocol successfully seized the line and resolved glare. The call advances to the placing call state. The call remains in this state until an event occurs indicating a state change:

This event...	Indicates that...
NCCEVN_CALL_PROCEEDING	The switch accepted the call setup request and is in the process of ringing the receiving end. The call is now in the proceeding call state.
NCCEVN_CALL_CONNECTED	A connection now exists between the calling parties. The call is now in the connected call state.
NCCEVN_CALL_DISCONNECTED	The remote party hung up. The call is now in the disconnected call state.

If the protocol supports the ability to overlap sending of digits, the application sends the remaining digit strings with **nccSendDigits** while the call is in the placing call state. For more information, see Sending overlapped digits.

Monitoring the proceeding call state

When the switch has accepted the call set up request, and is in the process of ringing the receiving end, the application receives NCCEVN_CALL_PROCEEDING. Call progress analysis is begun. The call state changes to proceeding.

If the call meets other connection criteria specified in this mask, the application receives NCCEVN_CALL_CONNECTED, and the call enters the connected call state. If the call does not meet the connection criteria or meets the disconnectmask criteria, the application receives NCCEVN_CALL_DISCONNECTED, and the call proceeds to the disconnected call state. In either case, the event's value field contains the reason for connection or disconnection.

The application can receive NCCEVN_BILLING_INDICATION while in the proceeding call state. This event indicates that a billing indication arrived. Depending upon the protocol, this indication may be a billing pulse or some more complex indication such as ISDN or SS7. In some cases, the value field of this event contains the billing units charged. No state transition occurs with this event. The application can invoke **nccQueryCapability** to determine if the protocol supports billing indications.

The NCC.START.eventmask parameter controls whether two other events appear during the proceeding call state:

Event	Description
NCCEVN_REMOTE_ALERTING	Indicates that the remote end is ringing. Does not cause a state transition. NCCEVN_REMOTE_ALERTING is generated only if the NCC_REPORT_ALERTING bit is set in the NCC.START.eventmask parameter.
NCCEVN_REMOTE_ANSWERED	Generated at the first positive indication that the remote party has answered. For example, out-of-band signaling, voice, or modem tone. The event value field contains the indication type. No state transition occurs. NCCEVN_REMOTE_ANSWERED is generated only if the NCC_REPORT_ANSWERED bit is set.

For more information, see NCC API global parameters.

Receiving billing indications

In any outbound call state, the application may receive NCCEVN_BILLING_INDICATION, indicating that billing information arrived. Depending upon the protocol, this indication may be a billing pulse or some more complex indication such as ISDN or SS7. In some cases, the value field of this event contains the billing units charged. No state transition occurs with this event.

NCCEVN_BILLING_INDICATION is generated only if the NCC_REPORT_BILLING bit is set in the NCC.START.eventmask parameter.

6. Other basic call control information

Disconnecting calls

When a call is torn down, the application can process it in one of two ways:

- Disconnect the call
- Release the call

Disconnecting a call

When a call is disconnected, a connection no longer exists between the local party and the remote party, but the call remains in the disconnected call state.

You can disconnect a call in any of the following ways:

- The application invokes **nccDisconnectCall**.

  If the NCC_CAP_DISCONNECT_IN_ANY_STATE indicator is set in the capabilitymask returned by **nccQueryCapability**, the application can invoke this function in any call state. Otherwise, the function can be invoked from the outbound initiated, placing, and connected call states only.
- The remote party hangs up. This can occur in almost any call state.
- The call is inbound, and the application rejects the call.
- (ADIMGR NCC implementation only) The call is outbound, the remote party answers, and the call fails to meet the criteria specified by the NCC_X_ADI_PLACECALL_PARMS connectmask passed with **nccPlaceCall**, or meets the criteria specified by the disconnectmask.
- A call is successfully transferred using **nccTransferCall** or **nccAutomaticTransfer**. For more information, refer to Transferring calls.

After the call is disconnected, the NCC API generates NCCEVN_CALL_DISCONNECTED. The call is now in the disconnected state. It is no longer active.

In this state, no connection exists between the local party and remote party. However, the call still exists logically; for example, the application can still invoke **nccGetCallStatus** to get information about the call or send protocol-specific call messages with **nccSendCallMessage**. With some protocols, the call can still be placed on hold. The application can invoke **nccQueryCapability** to determine if the protocol supports this feature.

Releasing call resources

Once a call is in the disconnected state, the application can invoke **nccReleaseCall** to release resources associated with the call, and destroy the call handle. The call no longer exists in any state.

When the call is released, NCCEVN_CALL_RELEASED is passed to the application. All internal resources allocated to the call are released. After the application receives this event, no more DSP-related events are generated for this call handle. The application can no longer retrieve call status information using **nccGetCallStatus** or **nccGetExtendedCallStatus**.

An outbound call can also be released if a glare (call collision) situation is detected while the call is in outbound initiated call state. In this case, the event value field contains NCC_RELEASED_GLARE.

If no other calls exist on the line after a call is released, the line state falls back to idle.

Protocol capabilities

Different protocols support different call control features. After a protocol is started, the application can query it to determine the features it supports. This information is important for an application supporting multiple protocols because it determines which function calls are valid, and which events the application can expect to receive.

To retrieve protocol capability information, the application calls **nccQueryCapability**. This synchronous function returns a capabilitymask, where each bit in the mask indicates if the protocol supports a particular capability. The following table describes each bit in the capabilitymask:

Mnemonic	If the bit is set...
NCC_CAP_ACCEPT_CALL	The protocol supports the accepting call state in the inbound call control state machine. The application can invoke **nccAcceptCall**.
NCC_CAP_SET_BILLING	The application can set billing parameters using **nccSetBilling**.
NCC_CAP_OVERLAPPED_SENDING	The protocol supports overlapped sending of digits, using **nccSendDigits**.
NCC_CAP_HOLD_CALL	The protocol allows calls to be placed on hold using **nccHoldCall**. Calls can then be retrieved using **nccRetrieveCall**.
NCC_CAP_SUPERVISED_TRANSFER	The protocol supports supervised call transfer. The application can invoke **nccTransferCall**.
NCC_CAP_AUTOMATIC_TRANSFER	The protocol supports blind call transfer. The application can invoke **nccAutomaticTransfer**.
NCC_CAP_TWOCHANNEL_TRANSFER	The protocol supports two channel call transfer. The application can invoke **nccTransferCall**.
NCC_CAP_EXTENDED_CALL_STATUS	The protocol supports the ability to get protocol-specific call status information using **nccGetExtendedCallStatus**.
NCC_CAP_SEND_CALL_MESSAGE	The protocol supports sending of a protocol-specific call message using **nccSendCallMessage**.
NCC_CAP_SEND_LINE_MESSAGE	The protocol supports sending of a protocol-specific line message using **nccSendLineMessage**.

Mnemonic	If the bit is set...
NCC_CAP_HOLD_IN_ANY_STATE	The protocol allows a call to be placed on hold in any state. By default, a call can be placed on hold only if it is in the connected state.
NCC_CAP_DISCONNECT_IN_ANY_STATE	The application can call **nccDisconnectCall** regardless of the call state. If this bit is not set, **nccDisconnectCall** can only be invoked in the connected or placing call states.
NCC_CAP_MEDIA_IN_SETUP	The protocol contains embedded media capability. The application can: • Use play ring and user audio modes when invoking **nccAcceptCall**. • Use play ring, play busy, play reorder, and play user audio modes when invoking **nccRejectCall**.
NCC_CAP_CALLER_ID	The protocol supports caller ID. Caller ID information is returned in the ***callingaddr*** field in the NCC_CALL_STATUS structure.

If the capabilities of a protocol change, the application receives NCCEVN_CAPABILITY_UPDATE. The application can then invoke **nccQueryCapability** to determine the changes.

The connected call state

In the connected call state, a connection exists between the calling parties. The application can now invoke other telephony services suitable for completing its tasks (for example, DTMF, MF, playing/recording) until a disconnect occurs. The application can also transfer the call as described in Transferring calls.

The call reaches the connected call state when one of the following occurs:

- If the call is inbound, it is answered by the application using **nccAnswerCall**.
- If the call is outbound, the remote party answers or is alerting, and the call meets the criteria specified by the NCC_PLACECALL_PARMS connectmask passed with **nccPlaceCall**.

When the call reaches the connected call state, NCCEVN_CALL_CONNECTED is returned to the application. The event value field contains the reason for connection. The call remains in this state until the application calls a function that changes the state, or receives an event indicating a state change.

The application can receive NCCEVN_BILLING_INDICATION while in the connected call state. This event indicates that a billing indication has arrived. Depending upon the protocol, this indication can be a billing pulse or some more complex indication such as ISDN or SS7. In some cases, the value field of this event contains the billing units charged. No state transition occurs with this event. The application can call **nccQueryCapability** to determine if the protocol supports billing indications.

The NCC.START.eventmask bit controls whether two other events appear during the connected call state. These events are only relevant if the call is outbound, the NCC_CON_ON_CALL_PROCEEDING bit is set in the NCC_PLACECALL_PARMS connectmask passed with **nccPlaceCall**, and the remote end has not yet answered:

Event	Description
NCCEVN_REMOTE_ALERTING	Indicates that the remote end is ringing. Does not cause a state transition.
NCCEVN_REMOTE_ANSWERED	Generated at the first positive indication that the remote party has answered; for example, out-of-band signaling, voice, or modem tone. The event value field contains the indication type. No state transition occurs.

For more information, refer to NCC API global parameters.

7. Advanced call control

Holding calls

The NCC API supports the ability to place a call on hold, and to retrieve a held call. Placing a call on hold frees the NCC API call control resources to establish another call connection.

Placing a call on hold does not change the call state. However, a call on hold is no longer active. Since there are no active calls currently on the line, the line state changes to idle.

Some protocols do not support call hold/retrieve. The NCC_CAP_HOLD_CALL bit in the capabilitymask returned by **nccQueryCapability** indicates if the protocol supports call hold/retrieve.

Placing a call on hold

To place a call on hold, the application invokes **nccHoldCall** with the handle of the call. When the application receives NCCEVN_CALL_HELD, the call is on hold. There is no call state transition. However, the line state goes to idle.

A call can be put on hold only from the connected state unless the capability bit NCC_CAP_HOLD_IN_ANY_STATE is set in the capabilitymask. The capabilitymask is retrieved with **nccQueryCapability**. If the bit is set, the call can be put on hold from any state, and change states while on hold.

The application can perform media functions on the held call such as playing a user voice file or silence, using a NaturalAccess media API. Depending upon the capabilities of the protocol, the application can also transfer a held call.

Retrieving a held call

To retrieve a held call, the application invokes **nccRetrieveCall** with the handle of the call. When the application receives NCCEVN_CALL_RETRIEVED, the call is retrieved. There is no call state transition. However, the line state returns to active.

A held call can be retrieved only if it is in the connected state unless the capability bit NCC_CAP_HOLD_IN_ANY_STATE is set in the capabilitymask. The capabilitymask is retrieved with **nccQueryCapability**.

Note: Some protocols allow calls to be held and retrieved remotely. A call is held when an unsolicited NCCEVN_CALL_HELD event is received. The call is retrieved when an unsolicited NCCEVN_CALL_RETRIEVED event is received.

Transferring calls

The NCC API supports the ability to transfer calls from a first party to a second party through a switch such as private branch exchange (PBX), Centrex, or Centrex-like exchange. The following call transfer methods are supported:

- Supervised transfer using **nccTransferCall**. With this method, the application performs the transfer process. The calls to be connected are on the same line.

  For the SIP API, the calls must be on two different lines, because SIP supports only one call per line. See the *Dialogic® NaturalAccess™ SIP API Developer's Manual* for more information.

- Blind, or automatic transfer using **nccAutomaticTransfer**. With this method, the application requests the switch to perform the transfer.
- Two channel transfer using **nccTransferCall**. With this method, the application performs the transfer process as a supervised transfer. The calls to be connected are on different lines.

Some protocols do not support all transfer methods. Refer to Protocol capabilities for more information.

Supervised transfer

In a supervised transfer, the application controls the transfer process. Before the application can perform a supervised transfer,

- Both calls must be in the connected state.
- Only one call can be on hold.
- Both calls must be on the same line.

Note: To transfer calls on different lines, use two channel transfer.

To perform the transfer, the application invokes **nccTransferCall** with the handles of the two calls. If the transfer completes successfully, the application receives two NCCEVN_CALL_DISCONNECTED events, one for each call. The value field of each event contains NCC_DIS_TRANSFER. As the event is returned for each call, the call enters the disconnected state from the point of view of the application. The application invokes **nccReleaseCall** for each call handle to destroy the handle and release resources. NCCEVN_CALL_RELEASED indicates that a call was released.

If the transfer does not complete successfully, the application receives NCCEVN_CALL_DISCONNECTED, with protocol-specific values. See the protocol-specific documentation for details.

Blind (automatic) transfer

In a blind or automatic transfer, the application requests the switch such as PBX, Centrex, or Centrex-like switch to perform a call transfer.

- Only one party is in the connected state.
- The current call must not be on hold.

To perform the transfer, the application invokes **nccAutomaticTransfer** with the handle of the current call and the address (for example, the extension) of the party to transfer to. When invoking the function, the application specifies at what point the transfer is to take place:

Value	Transfer takes place...
NCC_TRANSFER_PROCEEDING	After the address is sent, and the switch has accepted the calls the request and the call is in proceeding call state.
NCC_TRANSFER_ALERTING	When the remote end is alerting.
NCC_TRANSFER_CONNECTED	When the second call reaches the connected state.

Blind (automatic) transfer follows this process:

Stage	Description
1	The function places the first party on hold. When this occurs, NCCEVN_CALL_HELD is returned to the application.
2	The function then attempts to place a second call to the specified address.
3	If the placed call reaches the stage specified as the point at which the transfer is to take place, the transfer is performed. The application receives a single NCCEVN_CALL_DISCONNECTED event with the reason code NCC_DIS_TRANSFER, indicating that the transfer succeeded and the original call is now in the disconnected state from the point of view of the application. The application invokes **nccReleaseCall** to destroy the call handle and release resources.
4	If the transfer fails, the first call is retrieved from hold and remains in the connected state. NCCEVN_CALL_RETRIEVED is returned, with a value field containing one of the NCC_DIS_***xxx*** reason codes indicating why the transfer failed. For a list of possible NCC_DIS_***xxx*** values, see NCCEVN_CALL_DISCONNECTED.

Two channel transfer

Two channel transfer allows you to transfer calls that are on different lines.

- The calls do not need to be in a specific call state.
- A call identifier must exist for at least one of the calls. A call identifier internally identifies a call in a two channel transfer. If the bit parameter NCC_REPORT_CALLID in NCC.START.eventmask is set, NCCEVN_CALLID_AVAILABLE is sent to the application, and the application can issue a two channel transfer.

Refer to the protocol-specific documentation for more information on call identifiers, and on the appropriate time to issue a call transfer.

To perform the transfer, the application invokes **nccTransferCall** with the handles of the two calls. If the transfer is successful, the application receives two NCCEVN_CALL_DISCONNECTED events. The value field of one event contains NCC_DIS_TRANSFER. The value field of the other event contains a reason code indicating a remote hang up.

As the event is returned for each call, the call enters the disconnected state from the point of view of the application. The application invokes **nccReleaseCall** for each call handle to destroy the handle and release resources. NCCEVN_CALL_RELEASED indicates that a call was released.

If the transfer is not successful, the application receives NCCEVN_PROTOCOL_ERROR. The value field of this event contains NCC_PROTERR_TCT_FAILED. The size field contains the protocol-specific reason code. Call states for both calls remain unchanged when receiving this event.

Blocking calls

The application can block all incoming calls on a line and prevent placement of outbound calls on the line. While calls are being blocked, no call control events are generated for that line. Applications can also unblock calls that have been blocked.

Blocking a call

To block calls on a line, the application invokes **nccBlockCalls** with the line handle. **nccBlockCalls** can be invoked from any line state or call state. Incoming calls are blocked on the line when the line state is idle because there are no active calls on the line.

Note: Outbound-only trunks typically cannot be blocked. Blocking a line is a valid action only when the trunk is capable of receiving calls (inbound-only or two-way trunks).

When invoking **nccBlockCalls**, specify the method to use to block calls:

Method	Description
NCC_BLOCK_REJECTALL	Do not answer subsequent calls
NCC_BLOCK_OUT_OF_SERVICE	Place the line out of service

The NCC API waits until there are no active calls on the line. The line then enters the blocking state. The application receives NCCEVN_CALLS_BLOCKED. The line remains in the blocking state until **nccUnblockCalls** is invoked.

The application may receive an incoming call event after invoking **nccBlockCalls** and before receiving NCCEVN_CALLS_BLOCKED. The application must handle the call, answering or rejecting it as necessary. Calls are not blocked, and the application does not receive NCCEVN_CALLS_BLOCKED until there are no active calls.

Unblocking a call

To unblock calls on a line in blocking line state, the application invokes **nccUnblockCalls**, prompting the line state to change to idle. The application receives NCCEVN_CALLS_UNBLOCKED. At this point, the application can receive or place calls on the line.

The application can invoke **nccUnblockCalls** to cancel a previous **nccBlockCalls** invocation before the line state becomes idle. In this case, no events are received.

If the specified line is not in the blocking state, and no previously invoked **nccBlockCalls** is pending, **nccUnblockCalls** returns CTAERR_INVALID_STATE.

Getting status information

The following status information is available:

- Line status information, including the current state of the line and the name of the protocol running on the line.
- Call status information, including the current call state and the calling address (caller ID).
- Protocol-specific call status information.

Getting line status information

To get status information on a line, the application can invoke **nccGetLineStatus** with the line handle. This synchronous function can be called at any time, regardless of the line state or call state.

Line status information is returned in an NCC_LINE_STATUS structure. For details on this structure, refer to **nccGetLineStatus**.

Getting call status information

To get status information on a call, the application can invoke **nccGetCallStatus** with the call handle. This synchronous function can be invoked in any call state, as long as the line is in an active or an idle line state.

Call status information is returned in an NCC_CALL_STATUS structure. For details on this structure, refer to **nccGetCallStatus**.

If the calling address information (caller ID) changes, the application receives NCCEVN_CALL_STATUS_UPDATE. This event can occur in any call state, as long as the line state is active or idle. The value field of the event contains reason code NCC_CALL_STATUS_CALLINGADDR. The application can then invoke **nccGetCallStatus** to get the updated information.

This event is generated only if the NCC_REPORT_STATUSINFO bit is set in the NCC.START.eventmask parameter. For more information, refer to NCC API global parameters.

Getting protocol-specific call status information

With certain protocols, the application can retrieve a separate structure containing protocol-specific call status information. The application can invoke **nccQueryCapability** to determine if the protocol supports this feature. Protocol-specific call status information may include billing rates and user-to-user information (UUI). To learn more about the structure specific to your protocol, see your protocol-specific documentation.

To retrieve a structure containing protocol-specific information, the application invokes **nccGetExtendedCallStatus**. This synchronous function can be invoked in any call state, as long as the line state is active or idle.

If protocol-specific call status information changes, the application receives NCCEVN_EXTENDED_CALL_STATUS_UPDATE. The application can then invoke **nccGetExtendedCallStatus** to get the updated information.

This event is generated only if the NCC_REPORT_STATUSINFO bit is set in the NCC.START.eventmask parameter. For more information, refer to NCC API global parameters.

Sending protocol-specific information

An application can send protocol-specific information with almost any function call. This information can include billing rate indications, user-to-user (UUI) information, or anything supported by the protocol. An application can also send protocol-specific line or call messages without invoking a call control function.

Sending protocol-specific information with a function call

Most NCC API functions include a void* pointer that the application can use to reference a protocol-specific parameter structure. In cases where default parameters exist for a function, the application can pass NULL to use the default settings. To learn more about using this pointer with your protocol, see the protocol-specific documentation.

The following functions include the void* pointer:

- **nccAcceptCall**
- **nccAnswerCall**
- **nccAutomaticTransfer**
  Two protocol-specific parameter structures can be passed with this function: one contains call placement parameters for the second call; the other contains call transfer parameters.
- **nccBlockCalls**
- **nccDisconnectCall**
- **nccHoldCall**
- **nccPlaceCall**
- **nccRejectCall**
- **nccReleaseCall**
- **nccRetrieveCall**
- **nccSendCallMessage**
- **nccSendDigits**
- **nccSendLineMessage**
- **nccStartProtocol**
- **nccStopProtocol**
- **nccTransferCall**
- **nccUnblockCalls**

Sending protocol-specific call and line messages

The NCC API provides two functions that serve as escape mechanisms for sending call and line messages:

Use this function...	To send a...	From this line state...
nccSendCallMessage	Call message	Active or idle
nccSendLineMessage	Line message	Any line state except uninitialized

Not all protocols support sending of call messages or line messages using these escape mechanisms. The application can call **nccQueryCapability** to determine if the protocol supports one and/or the other. The NCC_CAP_SEND_CALL_MESSAGE bit in the capabilitymask indicates if the application can use **nccSendCallMessage**. NCC_CAP_SEND_LINE_MESSAGE indicates if the application can use **nccSendLineMessage**.

The application cannot use these mechanisms to specify a call or line message that causes a call or line state change as a result of successful message delivery.

Cooperative call processing

NaturalAccess provides the capability for two or more applications to cooperate in processing a call. In this type of cooperative environment, one application can create a call object, and a second application can release it. Each application obtains its own call handles to use when sharing call objects.

A call object is created when an outbound call is placed with **nccPlaceCall**, or when an inbound call generates NCCEVN_SEIZURE_DETECTED. Applications use call handles to reference call objects.

For more information on shared objects, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

Obtaining call handles

An application can obtain call handles by

- Invoking **nccPlaceCall** to initiate an outbound call.
- Receiving any NCC event associated with a call. The objHd field contains the call handle.
- Invoking **ctaAttachObject** and passing the object descriptor.
- Specifying a sufficiently sized buffer when invoking **nccGetLineStatus**.

Releasing call handles

When a call enters the disconnected state, the application can invoke **nccReleaseCall** to release the call. NCCEVN_CALL_RELEASED is passed to the application and the call handle on the application side is destroyed. Function calls that are passed a released call handle or an uninitialized call handle return CTA_INVALID_HANDLE.

8. Function summary

Telephony protocol functions

Function	Synchronous/ Asynchronous	Description
nccStartProtocol	Asynchronous	Loads a trunk control program (TCP) and enables the use of functions requiring DSP resources.
nccStopProtocol	Asynchronous	Halts the TCP on the line, terminates functions on the line, and unloads the TCP.

Incoming call functions

Function	Synchronous/ Asynchronous	Description
nccAcceptCall	Asynchronous	Accepts a call.
nccAnswerCall	Asynchronous	Answers a call.
nccRejectCall	Asynchronous	Rejects a call.

Outbound call functions

Function	Synchronous/ Asynchronous	Description
nccPlaceCall	Asynchronous	Places a call.
nccSendDigits	Asynchronous	Sends digits in overlapped sending mode.

Call holding, retrieval, and transfer functions

Function	Synchronous/ Asynchronous	Description
nccAutomaticTransfer	Asynchronous	Performs a blind call transfer.
nccHoldCall	Asynchronous	Puts a call on hold.
nccRetrieveCall	Asynchronous	Retrieves a held call.
nccTransferCall	Asynchronous	Performs a supervised call transfer or a two channel call transfer.

Protocol-specific message functions

Function	Synchronous/ Asynchronous	Description
nccSendCallMessage	Protocol specific	Protocol-specific escape mechanism to send a message to a call.
nccSendLineMessage	Protocol specific	Protocol-specific escape mechanism to send a message to a line.

Call disconnect and release functions

Function	Synchronous/ Asynchronous	Description
nccDisconnectCall	Asynchronous	Disconnects a call.
nccReleaseCall	Asynchronous	Releases resources associated with a call in the disconnected state, and destroys the call handle.

Status and protocol capability functions

Function	Synchronous/ Asynchronous	Description
nccGetCallStatus	Synchronous	Retrieves call control status information for a call.
nccGetExtendedCallStatus	Synchronous	Retrieves protocol-specific call control status information for a call.
nccGetLineStatus	Synchronous	Retrieves status information for a line.
nccQueryCapability	Synchronous	Queries capabilities of the current protocol.

Call blocking functions

Function	Synchronous/ Asynchronous	Description
nccBlockCalls	Asynchronous	Blocks incoming calls.
nccUnblockCalls	Asynchronous	Releases blocking of incoming calls.

Billing functions

Function	Synchronous/ Asynchronous	Description
nccSetBilling	Asynchronous	Sets billing relative to an incoming call.

Repeat dial functions

Function	Synchronous/ Asynchronous	Description
rdaAttach	Asynchronous	Attaches the repeat dial library to NCC.
rdaDetach	Asynchronous	Detaches the repeat dial library from NCC.
rdaLogCall	Asynchronous	Logs a call attempt into the repeat dial database.
rdaRequestCall	Asynchronous	Queries the repeat dial database to determine if another call to a given address can be attempted.

NCC functions and line states

The following table lists which NCC API calls can be made from each of the states of the line state machine.

Note: Functions marked by an asterisk can be performed in the idle line state on a call that is held, if supported by the service implementation and protocol. The NCC_CAP_HOLD_IN_ANY_STATE indicator in the capabilitymask returned by **nccQueryCapability** indicates if the protocol allows calls to be held or retrieved from any state.

NCC function	Line state				
	Active	Blocking	Idle	Out of service	Uninitialized
nccAcceptCall (*)	X		X		
nccAnswerCall (*)	X		X		
nccAutomaticTransfer	X				
nccBlockCalls	X		X	X	
nccDisconnectCall (*)	X		X		
nccGetCallStatus	X		X		
nccGetExtendedCallStatus	X		X		
nccGetLineStatus	X	X	X	X	X
nccHoldCall	X				
nccPlaceCall	X		X		
nccQueryCapability	X	X	X	X	
nccRejectCall (*)	X		X		
nccReleaseCall	X	X	X	X	
nccRetrieveCall	X		X		
nccSendCallMessage (*)	X		X		
nccSendDigits (*)	X		X		
nccSendLineMessage	X	X	X	X	

NCC function	Line state				
	Active	Blocking	Idle	Out of service	Uninitialized
nccSetBilling (*)	X		X		
nccStartProtocol					X
nccStopProtocol	X	X	X	X	
nccTransferCall	X				
nccUnblockCalls	X	X	X	X	

NCC functions and capability masks

The following table lists the functions that are not supported by some protocols and the name of the bit in the capabilitymask returned by **nccQueryCapability** that indicates if the function is supported.

NCC function	Capability mask
nccAcceptCall	NCC_CAP_ACCEPT_CALL
nccAutomaticTransfer	NCC_CAP_AUTOMATIC_TRANSFER
nccGetExtendedCallStatus	NCC_CAP_EXTENDED_CALL_STATUS
nccHoldCall	NCC_CAP_HOLD_CALL
nccRetrieveCall	NCC_CAP_HOLD_CALL
nccSendCallMessage	NCC_CAP_SEND_CALL_MESSAGE
nccSendDigits	NCC_CAP_OVERLAPPED_SENDING
nccSendLineMessage	NCC_CAP_SEND_LINE_MESSAGE
nccSetBilling	NCC_CAP_SET_BILLING
nccTransferCall	NCC_CAP_SUPERVISED_TRANSFER or NCC_CAP_TWOCHANNEL_TRANSFER

For more information, refer to Protocol capabilities.

NCC functions and call states

The following table lists which NCC API calls can be made from each of the states of the call state machine.

Functions marked with an asterisk (*) can be called in the indicated states if they are supported by the service implementation and protocol. The NCC_CAP_HOLD_IN_ANY_STATE indicator in the capabilitymask returned by **nccQueryCapability** indicates if the protocol allows calls to be held or retrieved from any state.

NCC function	Call state										
	Accepting	Answering	Connected	Disconnected	Incoming	Outbound initiated	Placing	Proceeding	Receiving digits	Rejecting	Seizure
nccAcceptCall					X				X		
nccAnswerCall	X				X				X		
nccAutomaticTransfer			X								
nccDisconnectCall			X			X	X	X			
nccGetCallStatus	X	X	X	X	X	X	X	X	X	X	X
nccGetExtendedCallStatus	X	X	X	X	X	X	X	X	X	X	X
nccHoldCall (*)	X *	X *	X	X *	X *	X *	X *	X *	X *	X *	X *
nccRejectCall	X				X				X		
nccReleaseCall				X							
nccRetrieveCall (*)	X *	X *	X	X *	X *		X *	X *	X *	X *	X *
nccSendCallMessage	X	X	X	X	X	X	X	X	X	X	X
nccSendDigits							X				
nccSetBilling	X	X	X		X				X	X	X
nccTransferCall			X								

NCC functions and capability masks

The following table lists the functions that are not supported by some protocols, and the name of the bit in the capabilitymask returned by **nccQueryCapability** that indicates if the function is supported.

NCC function	Capability mask
nccAcceptCall	NCC_CAP_ACCEPT_CALL
nccAutomaticTransfer	NCC_CAP_AUTOMATIC_TRANSFER
nccGetExtendedCallStatus	NCC_CAP_EXTENDED_CALL_STATUS
nccHoldCall	NCC_CAP_HOLD_CALL
nccRetrieveCall	NCC_CAP_HOLD_CALL
nccSendCallMessage	NCC_CAP_SEND_CALL_MESSAGE
nccSendDigits	NCC_CAP_OVERLAPPED_SENDING
nccSetBilling	NCC_CAP_SET_BILLING
nccTransferCall	NCC_CAP_SUPERVISED_TRANSFER NCC_CAP_TWOCHANNEL_TRANSFER

For more information, refer to Protocol capabilities.

nccDisconnectCall can also be called in the following states when the NCC_CAP_DISCONNECT_IN_ANY_STATE indicator is set in the capabilitymask returned by **nccQueryCapability**:

- Accepting
- Answering
- Incoming
- Receiving digits
- Rejecting
- Seizure

nccTransferCall can also be called in the following states when conducting two channel transfer. The NCC_CAP_TWOCHANNEL_TRANSFER indicator in the capabilitymask returned by **nccQueryCapability** indicates if the protocol supports two channel transfer:

- Accepting
- Answering
- Incoming
- Outbound initiated
- Placing
- Proceeding
- Receiving digits
- Rejecting
- Seizure

9. Function reference

Using the function reference

This section provides an alphabetical reference to the NCC API functions. A prototype of each function is shown with the function description and details of all arguments and return values. A typical function description includes:

Prototype	The prototype is shown followed by a list of the function arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to the Alphabetical error summary for a list of all errors returned by NCC API functions.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Refer to the event reference in this manual for details for all NCC API events.

nccAcceptCall

Accepts a call without answering or rejecting it.

Prototype

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

Argument	Description
callhd	Handle of the incoming call.
mode	Mode to accept an incoming call. See Details.

Argument	Description
acceptparms	Pointer to a protocol-specific NaturalAccess call acceptance parameter structure. Specify NULL to use the default values. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***mode*** is invalid.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the appropriate state to invoke this function.
NCCERR_NOT_CAPABLE	Returned when the protocol does not support the accepting call state. Also returned if ***mode*** is NCC_ACCEPT_PLAY_RING or NCC_ACCEPT_USER_AUDIO and the protocol does not support the media capability.

Events

Event	Description
NCCEVN_ACCEPTING_CALL	The function completed successfully. The call entered the accepting state.
NCCEVN_CALL_DISCONNECTED	The remote party disconnected. The call entered the disconnected state.

Details

nccAcceptCall accepts a call without answering or rejecting it, allowing the application to perform media functions, such as playing a voice file, before connecting or rejecting the call.

Not all protocols support this function and the accepting state. The application can determine if the protocol supports this state by examining the NCC_CAP_ACCEPT_CALL bit in the capabilitymask returned by **nccQueryCapability**.

If the remote party disconnects while the call is in the accepting state, the application receives NCCEVN_CALL_DISCONNECTED. In this case, no NCCEVN_ACCEPTING_CALL event is delivered to the application.

The application can specify one of the following modes to accept an incoming call:

Mode	Description
NCC_ACCEPT_PLAY_SILENT	Do not play anything.
NCC_ACCEPT_PLAY_RING	Play a ring tone.
NCC_ACCEPT_USER_AUDIO	User will supply audio.

NCC_ACCEPT_PLAY_RING and NCC_ACCEPT_USER_AUDIO are not supported by all protocols. The application can determine if the protocol supports this capability by examining the NCC_CAP_MEDIA_IN_SETUP bit in the capabilitymask returned by **nccQueryCapability**. NCC_ACCEPT_PLAY_SILENT is the default accept mode supported by all protocols.

See also

nccAnswerCall, **nccRejectCall**

nccAnswerCall

Answers a call.

Prototype

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

Argument	Description
callhd	Handle of the incoming call.
num_rings	Number of rings to play before answering.
answerparms	Pointer to a protocol-specific NaturalAccess call answering parameter structure. Specify NULL to use the default values. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.

Return value	Description
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the appropriate state to invoke this function.

Events

Event	Description
NCCEVN_ANSWERING_CALL	The function completed successfully and the call has entered the answering state.
NCCEVN_CALL_CONNECTED	The call was connected. The call enters the connected state.
NCCEVN_CALL_DISCONNECTED	The remote party disconnected. The call enters the disconnected state.

Details

If **nccAnswerCall** is successful, the call enters the answering state. The application receives NCCEVN_ANSWERING_CALL. When the call is connected, the application receives NCCEVN_CALL_CONNECTED, and the call enters the connected state. If the remote party disconnects while the call is in the answering state, the application receives NCCEVN_CALL_DISCONNECTED.

See also

nccAcceptCall, **nccRejectCall**

nccAutomaticTransfer

Transfers a call on a PBX, Centrex, or Centrex-like line.

Prototype

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

Argument	Description
callhd	Handle of the call being transferred.
digitstr	Pointer to the address (the party's extension) of the party to which to transfer to. This value must be a NULL terminated string with a maximum of 32 digits.

Argument	Description
transferwhen	Determines when the call is to be transferred. See Blind (automatic) transfer for the transfer values.
transferparms	Pointer to a protocol-specific NaturalAccess call transfer parameter structure. Specify NULL to use the default values. See the protocol-specific documentation for details.
mgrcallparms	Pointer to a manager-specific NaturalAccess call placement parameter structure. Specify NULL to use the default values. For all NaturalAccess protocols, this is a pointer to an NCC_ADI_PLACECALL_PARMS structure; the default parameters are NCC.X.ADI_PLACECALL.
protcallparms	Pointer to a protocol-specific NaturalAccess parameter structure. Specify NULL to use the default values. For the CAS API, this is a pointer to an NCC_ADI_CAS_PARMS structure; the default parameters are NCC.X.ADI_CAS. For ISDN Software, this is a pointer to a PLACECALL_EXT structure; the default parameters are NCC.X.ADI_ISDN.PALCECALL_EXT. For the SIP API, this is a pointer to a SIP data buffer. See the *Dialogic® NaturalAccess™ SIP API Developer's Manual* for more information.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	The specified digit string is NULL, is not NULL terminated, or has more than 32 digits.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_ADDRESS_BLOCKED	Too many calls were placed to this address or a call was placed too recently.
NCCERR_INVALID_CALL_STATE	The call is not in the connected state or is held.

Return value	Description
NCCERR_NOT_CAPABLE	The protocol does not support the capability to transfer a call automatically.

Events

Event	Description
NCCEVN_CALL_HELD	The first call is held.
NCCEVN_CALL_RETRIEVED	The transfer failed. The event value field contains one of the NCC_DIS_***xxx*** reason codes indicating why the transfer failed. For a list of possible NCC_DIS_***xxx*** values, see NCCEVN_CALL_DISCONNECTED.
NCCEVN_CALL_DISCONNECTED	***callhd*** is in disconnected call state. Receipt of this event with reason code NCC_DIS_TRANSFER indicates successful completion of the automatic transfer. The application should now release this call handle with **nccReleaseCall**. Other protocol-specific errors or reasons for disconnecting may be reported. See your protocol-specific documentation for more details.

Details

nccAutomaticTransfer executes a blind transfer by placing a second call and completing call transfer. This function operates only when the first call handle is in the connected state or is on hold.

nccAutomaticTransfer works as follows:

Stage	Description
1	The function places the first party on hold and returns NCCEVN_CALL_HELD to the application.
2	The function places a second call to the specified address.
3	If the placed call reaches the transferwhen stage, the transfer is performed. NCCEVN_CALL_DISCONNECTED is returned, with reason NCC_DIS_TRANSFER, indicating that the transfer succeeded and the first call is in the disconnected call state from the point of view of the application. The application should now invoke nccReleaseCall to destroy the call handle and release resources.

Not all protocols support automatic call transfer. The application can determine if the protocol supports this method by examining the NCC_CAP_AUTOMATIC_TRANSFER bit in the capabilitymask returned by **nccQueryCapability**. Refer to Blind (automatic) transfer for more information on how automatic transfer works.

Note: International regulatory approvals require you to limit attempts by an automated dialer to call a given number. Limits are placed on both the number and the frequency of the call attempts. For more information, refer to Limiting call attempts by automatic dialers.

See also

nccTransferCall

nccBlockCalls

Blocks incoming calls.

Prototype

DWORD **nccBlockCalls** (CTAHD ***linehd***, unsigned ***blockmode***,
void ****blockparms***)

Argument	Description
linehd	Line handle used to open the NCC API instance.
blockmode	Method used to block calls.
blockparms	Pointer to a protocol-specific NaturalAccess call blocking parameter structure. Specify NULL to use default values. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified handle.
CTAERR_INVALID_STATE	The current line state does not allow this function to be called.
CTAERR_SVR_COMM	A server communication error occurred.

Events

Event	Description
NCCEVN_BLOCK_FAILED	The request to block calls failed. The event value field contains the reason. For a list of reason codes, see NCCEVN_BLOCK_FAILED.
NCCEVN_CALLS_BLOCKED	The block calls request completed successfully. The line state changed to blocking.

Details

nccBlockCalls blocks calls on the specified line handle. **nccBlockCalls** can be invoked from the active, idle, or out of service line states. However, incoming calls are not blocked until there are no calls on the line and the line has returned to the idle state.

The method used to block calls is specified with the function invocation. Two methods are valid:

Method	Description
NCC_BLOCK_REJECTALL	Do not answer subsequent calls
NCC_BLOCK_OUT_OF_SERVICE	Place the line out of service

When the line state changes to blocking, the application receives NCCEVN_CALLS_BLOCKED. The value returned with this event contains the chosen blocking method. The line remains in the blocking state until **nccUnblockCalls** is invoked.

The application may receive NCCEVN_INCOMING_CALL after invoking **nccBlockCalls** and before receiving NCCEVN_CALLS_BLOCKED. The application must accept or answer the incoming call. The application does not receive NCCEVN_CALLS_BLOCKED until it releases all calls.

If the blocking request fails, the application receives NCCEVN_BLOCK_FAILED. The line remains in the current state.

Use **nccUnBlockCalls** to cancel a blocking request initiated using **nccBlockCalls**. You can cancel a blocking request any time before NCCEVN_CALLS_BLOCKED is received.

nccDisconnectCall

Disconnects a call that is connected to the network.

Prototype

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

Argument	Description
callhd	Handle of the call being disconnected.
disconnectparms	Pointer to a protocol-specific NaturalAccess call disconnect parameter structure. Specify NULL to use default values. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.

Return value	Description
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is already in the disconnected state.

Events

Event	Description
NCCEVN_CALL_DISCONNECTED	The disconnect operation completed successfully and the call entered the disconnected state.

Details

Use **nccDisconnectCall** to disconnect a connected call or to abandon outbound call placement.

The NCC_CAP_DISCONNECT_IN_ANY_STATE bit in the capabilitymask returned by **nccQueryCapability** determines in which states **nccDisconnectCall** can be invoked. If the bit is set, **nccDisconnectCall** can be invoked in any call state except disconnected. If the bit is cleared, **nccDisconnectCall** can only be invoked in the connected call state, and in outbound call states.

A disconnected call is no longer active. If no active calls exist on a line, the line state changes to idle.

See also

nccReleaseCall

nccGetCallStatus

Retrieves the call control status and stores it in an NCC_CALL_STATUS structure.

Prototype

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

Argument	Description
callhd	Handle of the call for which you want to retrieve status information.

Argument	Description

callstatus	A pointer to the NCC_CALL_STATUS structure to receive the call status:
	`typedef struct`
	`{`
	`DWORD size; /* No of bytes written to by callstatus */`
	`DWORD state; /* Current call state */`
	`char calledaddr [NCC_MAX_DIGITS+1]; /* Called number address */`
	`char callingaddr[NCC_MAX_DIGITS+1]; /* Calling number address */`
	`char callingname[NCC_MAX_CALLING_NAME]; /* Calling name info */`
	`DWORD pendingcmd; /* Last command not ack'ed by board */`
	`DWORD held; /* Non--zero value when call is held */`
	`DWORD direction; /* Indicates inbound or outbound call */`
	`CTAHD linehd; /* Line handle on which call resides */`
	`} NCC_CALL_STATUS;`
	See the Details section for field descriptions.

size	The size (in bytes) of the user-supplied status information space.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	The status pointer is NULL.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_SVR_COMM	A server communication error occurred.

Details

nccGetCallStatus retrieves the status of a specified call. When the function returns, the NCC_CALL_STATUS size field contains the actual number of bytes written.

Caller ID data is written to the NCC_CALL_STATUS structure. Applications that require caller ID data can check the calling party's ID by invoking **nccGetCallStatus** after NCCEVN_INCOMING_CALL is received, or by waiting for NCCEVN_CALL_STATUS_UPDATE, with a value code of NCC_CALL_STATUS_CALLINGADDR.

The NCC_CALL_STATUS structure contains the following fields:

Field	Description
size	Number of bytes written at the address pointed to by ***callstatus***.

Field	Description
state	Current call state. Possible state values are: • NCC_CALLSTATE_INVALID • NCC_CALLSTATE_SEIZURE • NCC_CALLSTATE_RECEIVING_DIGITS • NCC_CALLSTATE_INCOMING • NCC_CALLSTATE_ACCEPTING • NCC_CALLSTATE_ANSWERING • NCC_CALLSTATE_REJECTING • NCC_CALLSTATE_CONNECTED • NCC_CALLSTATE_DISCONNECTED • NCC_CALLSTATE_OUTBOUND_INITIATED • NCC_CALLSTATE_PLACING • NCC_CALLSTATE_PROCEEDING
calledaddr	For inbound calls, the address of the requested number, if provided (for example, DNIS, DID).
callingaddr	For inbound calls, the address of the caller, if provided (for example, caller ID, ANI).
callingname	For inbound calls, the name information of the caller, if provided.

Field	Description
pendingcmd	The last call control command issued that the board has not yet acknowledged. This field is set when a call control command is sent to the board, and cleared on the next event that corresponds the acknowledgment of the pending command. Possible pendingcmd values are: • (0) No command pending. • NCC_PENDINGCMD_ACCEPT_CALL • NCC_PENDINGCMD_ANSWER_CALL • NCC_PENDINGCMD_AUTOMATIC_TRANSFER • NCC_PENDINGCMD_DISCONNECT_CALL • NCC_PENDINGCMD_HOLD_CALL • NCC_PENDINGCMD_PLACE_CALL • NCC_PENDINGCMD_REJECT_CALL • NCC_PENDINGCMD_RETRIEVE_CALL • NCC_PENDINGCMD_SET_BILLING • NCC_PENDINGCMD_TRANSFER_CALL • NCC_PENDINGCMD_RELEASE_CALL
held	Set to non-zero value when a call is held.
direction	Indicates inbound or outbound call. Possible values are: • NCC_CALL_INBOUND • NCC_CALL_OUTBOUND
linehd	Line (context) handle on which the call resides.

See also

nccGetExtendedCallStatus, **nccGetLineStatus**

nccGetExtendedCallStatus

Retrieves protocol-specific status information for a call.

Prototype

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

Argument	Description
callhd	Handle of the call for which you want to retrieve status information.

Argument	Description
extendedcallstatus	A pointer to a protocol- and implementation-specific parameter structure. See the protocol-specific documentation for details.
size	The size (in bytes) of the application-supplied status information space.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***extendedcallstatus*** pointer is NULL.
CTAERR_BAD_SIZE	***size*** is smaller than the size of a DWORD.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_NOT_CAPABLE	The protocol does not support the ability to get extended call status information. The application can determine if the protocol supports this feature by examining the NCC_CAP_EXTENDED_CALL_STATUS bit in the capabilitymask returned by **nccQueryCapability**.

See also

nccGetCallStatus, **nccGetLineStatus**

nccGetLineStatus

Retrieves a snapshot of the port status and stores it in an NCC_LINE_STATUS structure.

Prototype

DWORD **nccGetLineStatus** (CTAHD ***linehd***, NCC_LINE_STATUS ****linestatus***, unsigned ***linestatussize***, NCC_CALLHD ***callhd***[], unsigned ***callhdsize***)

Argument	Description
linehd	Line handle used to open the NCC API instance.

Argument	Description

linestatus	A pointer to the NCC_LINE_STATUS structure to receive the information:
	`typedef struct`
	`{`
	`  DWORD size;  /* No of bytes written to by linestatus  */`
	`  DWORD state;  /* Current state of line  */`
	`  DWORD pendingcmd;  /* Last command not ack'd by board  */`
	`  char protocol[NCC_MAX_PNAME+1]; /* Array of protocols on line  */`
	`  CTA_MVIP_ADDR port;  /* MVIP address of port  */`
	`  unsigned numcallhd;  /* Number of unreleased call handles  */`
	`} NCC_LINE_STATUS;`
	See the Details section for field descriptions.

linestatussize	The amount of memory available at status. This value must be large enough to receive the NCC_LINE_STATUS size return value.
callhd	An array of call handles to be set to the values of calls on this line that have not been released. Set this value to NULL if you do not want to get a list of all the call handles on the line.
callhdsize	The size in bytes of the ***callhd*** array. This value is ignored if ***callhd*** is NULL.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	An argument is invalid or not usable.
CTAERR_BAD_SIZE	A size argument is invalid.
CTAERR_INVALID_HANDLE	The protocol is not running on the specified handle.
CTAERR_SVR_COMM	A server communication error occurred.

Details

nccGetLineStatus retrieves the status of the specified line. If the application provides a buffer that is large enough, this function retrieves a list of call handles existing on the line. Upon returning from the function, the NCC_LINE_STATUS size field contains the actual number of bytes written.

The NCC_LINE_STATUS structure contains the following fields:

Field	Description
size	Number of bytes written at the address pointed to by ***linestatus***.

Field	Description
state	Current line state. Can be any of the following: • NCC_LINESTATE_UNINITIALIZED • NCC_LINESTATE_IDLE • NCC_LINESTATE_BLOCKING • NCC_LINESTATE_OUT_OF_SERVICE • NCC_LINESTATE_ACTIVE
pendingcmd	The last line command issued that the board has not yet acknowledged. This field is set when a line control command is sent to the board, and cleared on the next event that corresponds to the acknowledgment of the pending command. Possible values are: • (0) No command pending. • NCC_PENDINGCMD_BLOCK_CALLS • NCC_PENDINGCMD_UNBLOCK_CALLS • NCC_PENDINGCMD_START_PROTOCOL • NCC_PENDINGCMD_STOP_PROTOCOL
protocol	Arrays containing names of protocols running on this line.
port	MVIP address of call control resource (port), if needed. Specified with **ctaOpenServices**.
numcallhd	Number of unreleased call handles, starting from index 0.

See also

nccGetCallStatus, **nccGetExtendedCallStatus**

nccHoldCall

Places a call on hold.

Prototype

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

Argument	Description
callhd	Handle of the call being placed on hold.
holdparms	Pointer to a protocol-specific NaturalAccess call hold parameter structure. Specify NULL to use default values. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the connected state. The protocol requires a call to be in the connected state before it can be put on hold.
NCCERR_NOT_CAPABLE	The protocol does not support the capability of putting a call on hold.

Events

Event	Description
NCCEVN_CALL_HELD	The call was successfully put on hold. The line is in an idle state.
NCCEVN_HOLD_REJECTED	The request to put a call on hold was rejected. The value field contains information on why the request was rejected. If the NCC_CAP_HOLD_IN_ANY_STATE bit is set in the capabilitymask returned by **nccQueryCapability**, depending upon what state the call is in, you may receive other events that are valid in that state. Refer to the event/call state chart in NCC events and call states for details.

Details

nccHoldCall puts a call on hold. There is no call state transition. A call on hold is no longer active. The line state changes to idle since there are no active calls currently on the line.

The call is on hold only after the application receives an NCCEVN_CALL_HELD event. The held field in the NCC_CALL_STATUS structure is set to a non-zero value.

Use **nccRetrieveCall** to release a call from being held.

Some protocols do not support the capability to put a call on hold. The application can determine if the protocol supports call hold/retrieve by examining the NCC_CAP_HOLD_CALL bit in the capabilitymask returned by **nccQueryCapability**.

Some protocols allow a call to be put on hold only from the connected state. The application can determine whether or not a call can be put on hold from any state by examining the NCC_CAP_HOLD_IN_ANY_STATE bit in the capabilitymask returned by **nccQueryCapability**.

The application can perform media functions on a held call, for example, play user voice file or play silence, using a NaturalAccess media service.

See also

nccTransferCall

nccPlaceCall

Places a call.

Prototype

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

Argument	Description
linehd	Line handle used to open the NCC instance.
calledaddr	Pointer to the address of the remote party. Must be a NULL terminated string with a maximum of 32 digits (NCC_MAX_DIGITS).
callingaddr	Pointer to the address of the local party. Must be a NULL terminated string with a maximum of 32 digits (NCC_MAX_DIGITS). Can be NULL if the protocol implementation allows.

mgrcallparms	For all NaturalAccess protocols, this is interpreted as a pointer to an NCC_ADI_PLACECALL_PARMS structure. The default parameters are NCC.X.ADI_PLACECALL.
	`typedef struct`
	`{`
	`DWORD size ;  /* size of this structure */`
	`DWORD connectmask;  /* events that transition to connected */`
	`DWORD disconnectmask;  /* events that transition to disconnected */`
	`ADI_CALLPROG_PARMS callprog;/* call progress analysis parameters */`
	`} NCC_ADI_PLACECALL_PARMS;`
	Refer to NCC.X.ADI_PLACECALL and to the *nccadi.h* file for more information.

protcallparms	Pointer to a protocol-specific NaturalAccess call placement parameter structure. Specify NULL to use default values. For the CAS API, this is a pointer to an NCC_ADI_CAS_PARMS structure; the default parameters are NCC.X.ADI_CAS. For ISDN Software, this is a pointer to a PLACECALL_EXT structure; the default parameters are NCC.X.ADI_ISDN.PALCECALL_EXT. For the SIP API, this is a pointer to a SIP data buffer. See the *Dialogic® NaturalAccess™ SIP API Developer's Manual* for more information.

Argument	Description
callhd	Pointer to NCC_CALLHD. NCC assigns a valid call handle to a successfully initialized outbound call. This call handle is used in other call-related functions, such as **nccHoldCall** or **nccReleaseCall**.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	The ***calledaddr*** string is NULL, is not NULL terminated, or has more than 32 digits. The ***callingaddr*** string is not NULL terminated or has more than 32 digits.
CTAERR_INVALID_HANDLE	The NCC API is not opened on this context handle.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_ADDRESS_BLOCKED	Too many calls were placed to this address, or a call was placed too recently.

Events

Event	Description
NCCEVN_PLACING_CALL	The call placement operation completed successfully and the call entered the placing state.
NCCEVN_CALL_RELEASED	An incoming call was detected during call placement. The TCP released the outbound call. The event value field contains NCC_RELEASED_GLARE.
NCCEVN_CALL_PROCEEDING	The switch accepted the call setup. The receiving side is being rung. The call enters the proceeding state.
NCCEVN_REMOTE_ALERTING	The remote end is in an alerting state (is ringing). This event is generated only if the NCC_REPORT_ALERTING bit is set in the NCC.START.eventmask parameter. For more information, refer to NCC API global parameters.

Event	Description
NCCEVN_REMOTE_ANSWERED	This event is generated at the first positive indication that the remote party has answered (for example, out-of-band signaling, voice, or modem tone). The event value field contains the indication type. This event is generated only if the NCC_REPORT_ANSWERED bit is set in the NCC.START.eventmask parameter. For more information, refer to NCC API global parameters.
NCCEVN_CALL_CONNECTED	Both parties are now connected. The call enters the connected state.

Details

calledaddr and ***callingaddr*** can be digit strings, IP addresses, or endpoint names, depending on the protocol.

If the application receives NCCEVN_CALL_RELEASED with an NCC_RELEASED_GLARE reason code instead of NCCEVN_PLACING_CALL, the line was seized for an incoming call before glare was resolved and the TCP has released the outgoing call. The application should immediately abandon outbound call placement and handle the incoming call. For information about managing call progress, refer to the *Dialogic® NaturalAccess™ Alliance Device Interface API Developer's Manual*.

The application can obtain protocol-specific information concerning the call, such as the B-channel location of a wireless protocol implementation, by invoking **nccGetExtendedCallStatus**.

Signaling protocols such as ISDN may support overlapped sending of digits. Such signaling may supply a partial dialing sequence when invoking **nccPlaceCall**, and use **nccSendDigits** to complete delivery of the remaining dialing sequence, in one or more chunks. The final character of each partial digit string should be a ~. When all digits are sent, the digit string specified in **nccSendDigits** should not include the final ~ character.

The application can determine whether or not a protocol supports overlapped sending of digits by examining the NCC_CAP_OVERLAPPED_SENDING bit in the capabilitymask returned by **nccQueryCapability**. For more information about overlapped sending of digits, see Sending overlapped digits.

Note: International regulatory approvals require you to limit attempts by an automated dialer to call a given number. Limits are placed on both the number and the frequency of the call attempts. For more information, refer to Limiting call attempts by automatic dialers.

nccQueryCapability

Queries the capabilities of the signaling protocol.

Prototype

DWORD **nccQueryCapability** (CTAHD ***linehd***, NCC_PROT_CAP ****protcap***, unsigned ***size***)

Argument	Description
linehd	Context handle used to open the NCC API instance.

protcap	A pointer to an NCC_PROT_CAP structure to hold returned protocol capability information.
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD capabilitymask;`
	`} NCC_PROT_CAP;`
	See the Details section for field descriptions.

size	The size of NCC_PROT_CAP structure.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified handle.
CTAERR_INVALID_STATE	The line ***linehd*** is in uninitialized state.
CTAERR_SVR_COMM	A server communication error occurred.

Details

nccQueryCapability queries the capabilities of the protocol on a specified line. Use this function to determine if the current protocol supports a given feature. **nccQueryCapability** returns an NCC_PROT_CAP structure containing capability information for the protocol.

The NCC_PROT_CAP structure contains the following fields:

Field	Description
size	Number of bytes written at the address pointed to by ***protcap***.
capabilitymask	A bit mask of features implemented by this protocol. See Protocol capabilities for a list of capabilities.

nccRejectCall

Rejects an incoming call.

Prototype

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

Argument	Description
callhd	Handle of the incoming call.
method	Method to be used to reject the call.
rejectparms	Pointer to a protocol-specific NaturalAccess reject call parameter structure. For a protocol-independent application using default parameters, pass NULL. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the appropriate state to invoke this function.
NCCERR_NOT_CAPABLE	Returned when NCC_REJECT_USER_AUDIO is specified when the protocol in use does not support the media capability. The application can determine whether the protocol supports this capability by examining the NCC_CAP_MEDIA_IN_SETUP bit in the capabilitymask returned by **nccQueryCapability**.

Events

Event	Description
NCCEVN_REJECTING_CALL	The reject call operation completed successfully and the call entered the rejecting state.
NCCEVN_CALL_DISCONNECTED	The remote party disconnected.

Details

After receiving an NCCEVN_INCOMING_CALL event, the application must invoke **nccAnswerCall**, **nccAcceptCall**, or **nccRejectCall**.

nccRejectCall causes the protocol to reject the incoming call using the method specified by ***method***. Valid methods are:

Method	Description
NCC_REJECT_PLAY_REORDER	Play reorder tone.
NCC_REJECT_PLAY_BUSY	Play busy tone.
NCC_REJECT_PLAY_RINGTONE	Play ring tone.
NCC_REJECT_USER_AUDIO	User will supply audio. This method is valid only if the NCC_CAP_MEDIA_IN_SETUP bit in the capabilitymask returned using **nccQueryCapability** is set.

When a call is rejected, the application receives NCCEVN_REJECTING_CALL.

To use the default rejection behavior for the current protocol, set ***method*** to 0.

If the remote party disconnects while the application is rejecting the call, the application receives NCCEVN_CALL_DISCONNECTED and not NCCEVN_REJECTING_CALL.

nccReleaseCall

Releases resources associated with a call in the disconnected state and destroys the call handle.

Prototype

DWORD **nccReleaseCall** (NCC_CALLHD ***callhd***, void ****releaseparms***)

Argument	Description
callhd	Handle of the call being released.
releaseparms	Pointer to protocol-specific NaturalAccess call release parameter structure. For a protocol-independent application using default parameters, pass NULL. See your protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is in uninitialized state.

Return value	Description
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the disconnected state.

Events

Event	Description
NCCEVN_CALL_RELEASED	Generated after the protocol has performed the network procedures for releasing the call. All internal resources allocated to the call are released. After the application receives this event, no more DSP-related events are generated for this call handle.

Details

nccReleaseCall can only be called when a call is in the disconnected state. When a call is released, it is completely destroyed. The application can no longer retrieve call status information.

See also

nccDisconnectCall

nccRetrieveCall

Retrieves a held call.

Prototype

DWORD **nccRetrieveCall** (NCC_CALLHD ***callhd***, void ****retrieveparms***)

Argument	Description
callhd	Handle of the call being retrieved.
retrieveparms	Pointer to a protocol-specific NaturalAccess call retrieve parameter structure. Protocol-independent applications can use default parameters by passing NULL for this argument. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.

Return value	Description
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the connected state. The protocol requires a call to be in the connected state before it can be put on hold or retrieved.

Events

Event	Description
NCCEVN_ CALL_RETRIEVED	The call was successfully retrieved.
NCCEVN_RETRIEVE_REJECTED	The request to retrieve a call was rejected. If the NCC_CAP_HOLD_IN_ANY_STATE bit is set in the capabilitymask returned by **nccQueryCapability**, depending upon what state the call is in, you may receive other events that are valid in that state. Refer to the event/call state chart in NCC events and call states for details.

Details

nccRetrieveCall retrieves a call previously placed on hold using **nccHoldCall**. The call becomes active again. The line state changes to active.

Some protocols do not support the capability to hold or retrieve calls. The application can determine if the protocol supports call hold or call retrieval by examining the NCC_CAP_HOLD_CALL bit in the capabilitymask returned by **nccQueryCapability**.

See also

nccTransferCall

nccSendCallMessage

Sends a protocol-specific call message to a protocol.

Prototype

DWORD **nccSendCallMessage** (NCC_CALLHD ***callhd***, void ****message***, unsigned ****size***)

Argument	Description
callhd	Handle for the call about which a message should be sent.
message	Pointer to buffer containing the protocol-specific call message.

Argument	Description
size	Size of the message buffer.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***message*** is NULL.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_NOT_CAPABLE	The protocol does not support the capability to send a call message.

Events

A protocol-specific event may be returned depending upon the protocol implementation.

Details

nccSendCallMessage sends a protocol-specific call message to a protocol. The protocol defines the allowed protocol-specific call messages that it can receive. Use of this function requires that the application be aware of which protocol is being used.

You cannot send a call message that causes a state change as a result of successful message delivery.

Some protocols do not support the capability to send a call message. The application can determine if the protocol supports this capability by examining the NCC_CAP_SEND_CALL_MESSAGE bit in the capabilitymask returned by **nccQueryCapability**.

Refer to your protocol-specific documentation for more details on the use of this function.

See also

nccSendLineMessage

nccSendDigits

Continues the process of sending digits to place an outbound call (for protocols that support overlapped sending of digits).

Prototype

DWORD **nccSendDigits** (NCC_CALLHD ***callhd***, char ****digits***, void ****senddigitparms***)

Argument	Description
callhd	Handle of the call for which to send digits.

Argument	Description
digits	Digit string to send. A digit string ending with a ~ indicates that more digit strings will be sent.
senddigitsparms	Pointer to a protocol-specific NaturalAccess **nccSendDigits** parameter structure. For protocol-independent applications, set this argument to NULL to use the default parameters.

Return values

Return value	Description
SUCCESS	The digit string was successfully sent.
CTAERR_BAD_ARGUMENT	The ***digits*** string is NULL.
CTAERR_INVALID_HANDLE	The call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the placing state.
NCCERR_NOT_CAPABLE	The protocol does not support the overlapped sending capability.

Details

Signaling protocols (for example, ISDN) may support the overlapped sending of digits. Such signaling may supply a partial dialing sequence in the initial **nccPlaceCall** invocation, and then use **nccSendDigits** to complete delivery of the remaining dialing sequence. The final character of each partial digit string should be a ~. When all digits are sent, the digit string specified in **nccSendDigits** should not include the final ~ character. This signals to the protocol that all digits have arrived. See the protocol-specific documentation for details.

Depending on the protocol, ***calledaddr*** and ***callingaddr*** can be digit strings, IP addresses, or endpoint names.

If the application receives an incoming call while it is placing a call and sending digits, it must abandon call placement and process the incoming call.

The application can determine whether the protocol supports the overlapped sending of digits by examining the NCC_CAP_OVERLAPPED_SENDING bit in the capabilitymask returned by **nccQueryCapability**.

The application can obtain protocol-specific information concerning the call, such as the B-channel location of a wireless protocol implementation, by invoking **nccGetExtendedCallStatus**.

nccSendLineMessage

Sends a protocol-specific line message to a protocol.

Prototype

DWORD **nccSendLineMessage** (CTAHD ***linehd***, void ****message***, unsigned ***size***)

Argument	Description
linehd	Line handle used to open the NCC API instance.
message	Pointer to a buffer containing the protocol-specific line message.
size	Size of the message buffer.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***message*** is NULL.
CTAERR_INVALID_HANDLE	The protocol is not running on the specified handle.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_NOT_CAPABLE	The protocol does not support the capability to send a line message.

Events

A protocol-specific event may be returned depending upon the protocol implementation.

Details

nccSendLineMessage sends a protocol-specific line message to a protocol. The protocol defines the allowed protocol-specific line messages that it can receive. The application must be aware of which protocol is being used.

You cannot send a line message that causes a state change as a result of successful message delivery.

Some protocols do not support the capability to send a line message. The application can determine if the protocol supports this capability by examining the NCC_CAP_SEND_LINE_MESSAGE bit in the capabilitymask returned by **nccQueryCapability**.

Refer to the protocol-specific documentation for more details on the use of this function.

See also

nccSendCallMessage

nccSetBilling

Sets billing information relative to an incoming call.

Prototype

DWORD **nccSetBilling** (NCC_CALLHD ***callhd***, unsigned ***rate*** void ****billingparms***)

Argument	Description
callhd	Handle of the incoming call.
rate	Billing rate (a country-dependent unit of cost).
billingparms	Pointer to a protocol-specific NaturalAccess billing parameter structure. For a protocol-independent application using default parameters, pass NULL. See your protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified handle**.**
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	The call is not in the appropriate state to invoke this function.
NCCERR_NOT_CAPABLE	The protocol does not support the capability of setting billing.

Events

Event	Description
NCCEVN_BILLING_SET	The billing information was successfully sent to the network. The value field contains one of the following: NCC_BILLINGSET_DEFAULT Billing rate set to default value NCC_BILLINGSET_FREE Billing rate free (billing rate) Billing rate value

Details

nccSetBilling sets billing information relative to an incoming call.

Some protocols do not support the capability of setting billing. The application can determine if the protocol supports this capability by examining the NCC_CAP_SET_BILLING bit in the capabilitymask returned by **nccQueryCapability**.

nccStartProtocol

Prepares an uninitialized line to be used by a protocol to accept and/or place calls.

Prototype

DWORD **nccStartProtocol** (CTAHD ***linehd***, char ****protname***, NCC_START_PARMS ****startparms,*** void ****mgrstartparms***, void ****protstartparms***)

Argument	Description
linehd	Line handle to initialize.
protname	Name of protocol to start on the line.

Argument	Description

startparms	Pointer to a NCC_START_PARMS structure containing generic NCC protocol-starting parameters. To use the default parameters, set this argument to NULL.
	`typedef struct`
	`{`
	`  DWORD size;`
	`  DWORD eventmask;`
	`  WORD  debugflag;`
	`  WORD  waitforPCtime;`
	`  WORD  overlappedreceiving;`
	`  struct`
	`  {`
	`  DWORD size;`
	`  WORD  dialtonefreq1;`
	`  WORD  dialtonefreq2;`
	`  WORD  dialtoneontime1;`
	`  WORD  dialtoneontime2;`
	`  WORD  dialtoneofftime1;`
	`  WORD  dialtoneofftime2;`
	`  WORD  dialtonelevel;`
	`  WORD  ringfreq1;`
	`  WORD  ringfreq2;`
	`  WORD  ringontime;`
	`  WORD  ringofftime1;`
	`  WORD  ringofftime2;`
	`  WORD  ringtonelevel;`
	`  WORD  busyfreq1;`
	`  WORD  busyfreq2;`
	`  WORD  busyontime;`
	`  WORD  busyofftime;`
	`  WORD  busytonelevel;`
	`  WORD  fastbusyfreq1;`
	`  WORD  fastbusyfreq2;`
	`  WORD  fastbusyontime;`
	`  WORD  fastbusyofftime;`
	`  WORD  fastbusytonelevel;`
	`  }  callproggenerate;`
	`} NCC_START_PARMS;`
	See NCC.START structure and NCC.START.callproggenerate parameters for complete field descriptions.

mgrstartparms	Pointer to a manager-specific NaturalAccess protocol-starting parameter structure. Protocol-independent applications can use the default parameters by setting this argument to NULL. For parameter descriptions for the ADI implementation of the NCC API, see NCC.X.ADI_START.
protostartparms	Pointer to a protocol-specific NaturalAccess protocol-starting parameter structure. Protocol-independent applications can use the default parameters by setting this argument to NULL.

Return values

Return value	Description
SUCCESS	

Return value	Description
CTAERR_BAD_ARGUMENT	***protname*** is NULL.
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified line.
CTAERR_INVALID_STATE	A protocol is already started on this line.
CTAERR_SVR_COMM	A server communication error occurred.

Events

Event	Description
NCCEVN_START_PROTOCOL_DONE	Acknowledges the application's attempt to start a protocol on a context (line handle). The event value field indicates if the protocol was started or not. For a list of reason codes, see NCCEVN_START_PROTOCOL_DONE.

Details

The event value field can also contain CTAERR_BOARD_ERROR, indicating that an error occurred on the board when the application attempted to start a protocol. If you configured the echo canceller, verify that the DSP file was downloaded to the board.

Refer to NCC API global parameters for default NCC_START_PARMS parameter settings.

See also

ctaOpenServices, **nccStopProtocol**

nccStopProtocol

Uninitializes an NCC line.

Prototype

DWORD **nccStopProtocol** (CTAHD ***linehd***, void* ***stopparms***)

Argument	Description
linehd	Line handle to uninitialize.
stopparms	Pointer to a protocol-specific NaturalAccess stop-protocol parameter structure. For a protocol-independent application using default parameters, pass NULL. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified line.
CTAERR_INVALID_STATE	No protocol is currently running on this line.
CTAERR_SVR_COMM	A server communication error occurred.

Events

Event	Description
NCCEVN_CALL_RELEASED	A call that was on the line was released before the protocol was stopped.
NCCEVN_STOP_PROTOCOL_DONE	The line state was successfully changed from idle to uninitialized. The line can no longer be used to accept and/or place calls.

Details

If **nccStopProtocol** succeeds, the line state becomes uninitialized. All calls on the line are automatically released before NCCEVN_STOP_PROTOCOL_DONE is returned.

See also

ctaOpenServices, **nccStartProtocol**

nccTransferCall

Performs a supervised transfer or a two channel transfer.

Prototype

DWORD **nccTransferCall** (NCC_CALLHD ***callhd1***, NCC_CALLHD ***callhd2***, void ****transferparms***)

Argument	Description
callhd1	Handle of a call to be transferred.
callhd2	Handle of a call to be transferred.

Argument	Description
transferparms	A pointer to a protocol-dependent NaturalAccess call transfer parameter structure. Protocol-independent applications can pass NULL for this argument to use the defaults. See the protocol-specific documentation for details. If you are doing a supervised transfer for SIP, set ***transferparms*** to SIP_SUPERVISED_TRANSFER.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	A call handle is not valid. It may have been released.
CTAERR_INVALID_STATE	The line is not in the appropriate state to invoke this function Returned only in supervised transfer.
CTAERR_SVR_COMM	A server communication error occurred.
NCCERR_INVALID_CALL_STATE	In supervised transfer, one call must be in the connected state and be on hold. The other call must be in the connected state and *not* be on hold. If these conditions are not satisfied, this error code is returned.
NCCERR_NO_CALLID	Could not find a call identifier for a two channel transfer.
NCCERR_NOT_CAPABLE	The protocol does not support the capability to perform a supervised transfer or a two channel transfer.

Events

Event	Description
NCCEVN_CALL_DISCONNECTED	In a supervised transfer, this event is received twice: once for each call. In a two channel transfer, this event is received twice if the transfer is successful.

Event	Description
NCCEVN_PROTOCOL_ERROR	In a two channel transfer, this event is received if call transfer fails. The value field contains NCC_PROTERR_TCT_FAILED. The size field contains the protocol-specific reason code. For more information, refer to the protocol-specific documentation.

Details

nccTransferCall performs a supervised call transfer or a two channel call transfer. For details, refer to Supervised transfer and Two channel transfer.

Note: Supervised call transfer is not supported in all variants. The application can determine if the protocol supports this state by examining the NCC_CAP_SUPERVISED_TRANSFER bit in the capabilitymask returned by **nccQueryCapability**. For QSIG, set the CC_USE_PATH_REPLACEMENT out calls behavior bit so that **nccTransferCall** invokes the path replacement supplementary service instead of the transfer service.

See also

nccAutomaticTransfer

nccUnblockCalls

Releases the block on incoming calls.

Prototype

DWORD **nccUnblockCalls** (CTAHD ***linehd***, void ****unblockparms***)

Argument	Description
linehd	Line handle used to open the NCC API instance.
unblockparms	Pointer to a protocol-specific NaturalAccess call unblocking parameter structure. Protocol-independent applications can pass NULL for this argument to use the defaults. See the protocol-specific documentation for details.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	The NCC API is not open on the specified handle.
CTAERR_INVALID_STATE	The specified line is not in the blocking state, and no previously invoked **nccBlockCalls** is pending.

Return value	Description
CTAERR_SVR_COMM	A server communication error occurred.

Events

Event	Description
NCCEVN_CALLS_UNBLOCKED	The unblock calls request completed successfully. The line state changed to idle.
NCCEVN_UNBLOCK_FAILED	The request to unblock calls failed. The event value field contains the reason. For a list of reason codes, see NCCEVN_UNBLOCK_FAILED.

Details

nccUnblockCalls cancels call blocking on the specified line handle. When the line state changes to idle, NCCEVN_CALLS_UNBLOCKED is generated. The application can then receive or place new calls.

Use **nccUnblockCalls** to cancel a previous **nccBlockCalls** invocation before the protocol is able to start blocking calls on the specified line. In this case, the application receives no events.

If the request fails, the application receives NCCEVN_UNBLOCK_FAILED. The line remains in the current state.

rdaAttach

Attaches the repeat dial library (*rda.dll* under Windows and *librda.so* under UNIX) to NCC.

Prototype

DWORD **rdaAttach** (void ***extparms**)

Argument	Description
extparms	Reserved for future use.

Return values

SUCCESS

Details

rdaAttach attaches the repeat dial library to NCC and initializes the repeat dial database. This function is invoked when the NCC API is opened.

See also

rdaDetach

rdaDetach

Detaches the repeat dial library from NCC.

Prototype

DWORD **rdaDetach** (void ***extparms**)

Argument	Description
extparms	Reserved for future use.

Return values

SUCCESS

Details

rdaDetach detaches the repeat dial library from NCC and shuts down the database if no other connections exist. This function is invoked when the NCC API is closed.

See also

rdaAttach

rdaLogCall

Logs a call attempt into the repeat dial database.

Prototype

DWORD **rdaLogCall** (char ***address**, void ***extparms**)

Argument	Description
address	Pointer to the address of the party being called.
extparms	Reserved for future use.

Return values

SUCCESS

Details

rdaLogCall logs a call attempt in the repeat dial database. This function is invoked when a call is in the proceeding state (the switch is ringing the dialed number).

See also

rdaRequestCall

rdaRequestCall

Queries the repeat dial database to determine if another call to a given address can be attempted.

Prototype

DWORD **rdaRequestCall** (char ***address**, void ***extparms**)

Argument	Description
address	Pointer to the address of the party being called.
extparms	Reserved for future use.

Return values

Return value	Description
SUCCESS	
NCCERR_ADDRESS_BLOCKED	Too many calls were placed to this address, or a call was placed too recently.

Details

rdaRequestCall queries the repeat dial database to determine if another call to a given address can be attempted. **rdaRequestCall** uses the rules defined in *rda.cfg* to determine how many call attempts will be made to a given address and the minimum interval between calls. This function is invoked during **nccPlaceCall** or **nccAutomaticTransfer**.

See also

rdaLogCall

10. Event summary

Protocol management events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_START_PROTOCOL_DONE	Transitional	Solicited	The application attempted to start a protocol on a context (line handle).
NCCEVN_STOP_PROTOCOL_DONE	Transitional	Solicited	The protocol stopped running on the context. The line is uninitialized.

Incoming call events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_ACCEPTING_CALL	Transitional	Solicited	The application accepted the call referenced by the call handle.
NCCEVN_ANSWERING_CALL	Transitional	Solicited	The application answered the call referenced by the call handle.
NCCEVN_CALL_CONNECTED	Transitional	Unsolicited	Both parties are connected.
NCCEVN_INCOMING_CALL	Transitional	Unsolicited	All information related to an incoming call was collected, and is now presented to the application.
NCCEVN_RECEIVED_DIGIT	Transitional	Unsolicited	Digits are being received in an overlapped fashion for an incoming call.
NCCEVN_REJECTING_CALL	Transitional	Solicited/ Unsolicited	A call is being rejected.

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_SEIZURE_DETECTED	Transitional	Unsolicited	The network seized the line.

Outbound call events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_CALL_CONNECTED	Transitional	Unsolicited	Both parties are connected.
NCCEVN_CALL_PROCEEDING	Transitional	Unsolicited	The switch accepted the call setup. The receiving side is being rung.
NCCEVN_PLACING_CALL	Transitional	Solicited	A call was placed successfully.
NCCEVN_REMOTE_ALERTING	Informational	Solicited	The remote end is ringing.
NCCEVN_REMOTE_ANSWERED	Informational	Solicited	The remote end is answering the call.

Call holding, retrieval, and transfer events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_CALL_HELD	Transitional	Solicited/ Unsolicited	A call was placed on hold by the application or by a remote party.
NCCEVN_CALL_RETRIEVED	Transitional	Solicited/ Unsolicited	A held call was retrieved by the application or by a remote party.
NCCEVN_HOLD_REJECTED	Informational	Solicited	A request to put a call on hold was rejected.
NCCEVN_CALLID_AVAILABLE	Informational	Unsolicited	A call identifier used to identify a call in two channel transfer is available.

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_RETRIEVE_REJECTED	Informational	Solicited	A retrieve call request was rejected.

Call disconnect and call release events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_CALL_DISCONNECTED	Transitional	Solicited/ Unsolicited	A call was disconnected by the application or by a remote party.
NCCEVN_CALL_RELEASED	Transitional	Solicited	The application released a call; NCC released all internal resources for the call.

Call status and protocol capability status events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_CALL_STATUS_UPDATE	Informational	Unsolicited	A call status changed.
NCCEVN_CAPABILITY_UPDATE	Informational	Unsolicited	Protocol capabilities changed.
NCCEVN_EXTENDED_CALL_STATUS_UPDATE	Informational	Unsolicited	Extended call status information changed.

Call blocking events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_BLOCK_FAILED	Informational	Solicited	A request to block calls on a line failed.
NCCEVN_CALLS_BLOCKED	Transitional	Solicited	The application blocked all calls on a line.

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_CALLS_UNBLOCKED	Transitional	Solicited	The line is no longer blocked.
NCCEVN_UNBLOCK_FAILED	Informational	Solicited	A request to unblock a line failed.

Call billing events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_BILLING_INDICATION	Informational	Unsolicited	A billing indication arrived.
NCCEVN_BILLING_SET	Informational	Solicited	A request to set the billing for a call is complete.

Line and protocol status events

Event	Informational / Transitional	Solicited / Unsolicited	Indicates that...
NCCEVN_LINE_IN_SERVICE	Transitional	Unsolicited	The network placed the line in service (was out of service).
NCCEVN_LINE_OUT_OF_SERVICE	Transitional	Unsolicited	The network placed the line out of service (was in service).
NCCEVN_PROTOCOL_ERROR	Informational	Solicited/ Unsolicited	A two channel call transfer failed or an error condition occurred. The call may be in an unusable state.
NCCEVN_PROTOCOL_EVENT	Informational	Solicited/ Unsolicited	A protocol-specific event occurred.

11. Event reference

Overview of the NCC API events

When performing call control functions, the NCC API processes two kinds of events: those that arrive from the network, and those that are generated in response to NCC API commands. The NCC API translates the network events into generic call control events. The names are defined in uppercase letters with an NCCEVN_ prefix. The network events fall into two classes: transitional and informational.

- Transitional events are generated when the call or line state changes. They are generated because the NCC API needs the application to choose an action or acknowledge that an application command is proceeding.
- Informational events do not change the line or call state. For example, the protocol error event, NCCEVN_PROTOCOL_ERROR, provides information about abnormalities on the line including false seizure, too many incoming digits, or premature answer while dialing.

Events are also classified as solicited or unsolicited.

- Solicited events are generated as a result of an application executing a command.
- Unsolicited events are the consequences of activities on the network.

Some NCC API events are solicited events in some circumstances, and unsolicited events in other circumstances. Some events are generated only if you enable them by setting them in the NCC.START.eventmask parameter. For more information about this parameter, see NCC API global parameters.

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;
```

This 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 function library. All NCC events are prefixed with NCCEVN_ . All NaturalAccess events are prefixed with CTAEVN_.
ctahd	Contains the context handle (line handle) returned from **ctaCreateContext**.

Field	Description
timestamp	Contains the time when the event was created in milliseconds.
userid	Contains the user-supplied id. This field is unaltered by NaturalAccess and facilitates asynchronous programming. Its purpose is to correlate a context with an application object/context when events occur.
size	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	This field 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	This is an event-specific value. This field can hold a reason code or an error code.
objHd	Contains the call handle, if the event concerns a particular call. If the event concerns the line and not a particular call, objHd is NULL.

NCC events and states

When a line or call is in a given state, any unsolicited event that is valid to be received in that state can be received regardless of any pending function calls. For example, assume an application calls **nccAnswerCall** for a call in the NCC_CALLSTATE_RECEIVING_DIGITS state. Any unsolicited events that can be received in the NCC_CALLSTATE_RECEIVING_DIGITS state are valid, even while **nccAnswerCall** is pending. You do not proceed into the NCCEVN_CALLSTATE_ANSWERING state until the NCCEVN_CALL_ANSWERED event is received.

Your application does not have to check for certain solicited events if the application has not invoked the corresponding function call. For instance, continuing the previous example, NCCEVN_ACCEPTING_CALL will not arrive because the function **nccAcceptCall** was not invoked. To determine if an event is solicited, unsolicited, or both, refer to the event details in the alphabetical event listing.

Some events are generated only if they are enabled by setting a parameter. For instance, if overlapped receiving is turned on (NCC.START.overlappedreceiving parameter = 1), the application receives NCCEVN_RECEIVED_DIGIT for calls on that line. For a discussion of how to use parameters to specify the events your application can handle, see the NCC.START.eventmask and NCC.START.overlappedreceiving parameter descriptions in NCC API global parameters.

Refer to NCC events and line states and NCC events and call states to determine which events your application may receive in a given line/call state.

NCC events and line states

The following table specifies standard NCC events that the application can expect to receive for a line when the line is in a particular line state. Other protocol-specific NCC events may be reported in most states. See the protocol-specific documentation for details.

Events marked with an asterisk (*****) can occur in the idle state if the NCC_CAP_HOLD_IN_ANY_STATE bit is set in the capabilitymask returned by **nccQueryCapability**.

NCC event	Line state				
	Active	Blocking	Idle	Out of service	Initialized
NCCEVN_ACCEPTING_CALL (*)	X		X *		
NCCEVN_ANSWERING_CALL (*)	X		X *		
NCCEVN_BILLING_INDICATION (*)	X		X *		
NCCEVN_BILLING_SET (*)	X		X *		
NCCEVN_BLOCK_FAILED			X		
NCCEVN_CALL_CONNECTED (*)	X		X *		
NCCEVN_CALL_DISCONNECTED (*)	X		X *		
NCCEVN_CALL_HELD	X				
NCCEVN_CALL_PROCEEDING (*)	X		X *		
NCCEVN_CALL_RELEASED	X	X	X	X	
NCCEVN_CALL_RETRIEVED			X		
NCCEVN_CALL_STATUS_UPDATE	X		X		
NCCEVN_CALLID_AVAILABLE (*)	X		X *		
NCCEVN_CALLS_BLOCKED			X		
NCCEVN_CALLS_UNBLOCKED		X			
NCCEVN_CAPABILITY_UPDATE	X	X	X	X	
NCCEVN_EXTENDED_CALL_STATUS_UPDATE	X		X		
NCCEVN_HOLD_REJECTED	X				

NCC event	Line state				
	Active	Blocking	Idle	Out of service	Initialized
NCCEVN_INCOMING_CALL (*)	X		X *		
NCCEVN_LINE_IN_SERVICE				X	
NCCEVN_LINE_OUT_OF_SERVICE			X		
NCCEVN_PLACING_CALL	X				
NCCEVN_PROTOCOL_ERROR	X	X	X	X	
NCCEVN_PROTOCOL_EVENT	X	X	X	X	
NCCEVN_RECEIVED_DIGIT (*)	X		X *		
NCCEVN_REJECTING_CALL (*)	X		X *		
NCCEVN_REMOTE_ALERTING (*)	X		X *		
NCCEVN_REMOTE_ANSWERED (*)	X		X *		
NCCEVN_RETRIEVE_REJECTED			X		
NCCEVN_SEIZURE_DETECTED	X		X		
NCCEVN_START_PROTOCOL_DONE					X
NCCEVN_STOP_PROTOCOL_DONE	X	X	X	X	
NCCEVN_UNBLOCK_FAILED		X			

NCC events and parameter control

The following table lists NCC events that are controlled by parameters. For more information about parameters, see NCC API global parameters.

NCC event	Controlled by this parameter...
NCCEVN_BILLING_INDICATION	NCC_REPORT_BILLING bit in NCC.START.eventmask

NCC event	Controlled by this parameter...
NCCEVN_CALL_STATUS_UPDATE	NCC_REPORT_STATUSINFO bit in NCC.START.eventmask
NCCEVN_CALLID_AVAILABLE	NCC_REPORT_CALLID bit in NCC.START.eventmask
NCCEVN_EXTENDED_CALL_STATUS_UPDATE	NCC_REPORT_STATUSINFO bit in NCC.START.eventmask
NCCEVN_RECEIVED_DIGIT	NCC.START.overlapped receiving
NCCEVN_REMOTE_ALERTING	NCC_REPORT_ALERTING bit in NCC.START.eventmask
NCCEVN_REMOTE_ANSWERED	NCC_REPORT_ANSWERED bit in NCC.START.eventmask

NCC events and call states

The following table specifies standard NCC events that the application can expect to receive for a call when the call is in a particular call state. Other protocol-specific NCC events may be reported in most states. See the protocol-specific documentation for details.

Events marked with an asterisk (*****) can occur in the indicated states if the NCC_CAP_HOLD_IN_ANY_STATE bit is set in the capabilitymask returned by **nccQueryCapability.**

NCC event	NCC call state										
	Accepting	Answering	Connected	Disconnected	Incoming	Outbound initiated	Placing	Proceeding	Receiving digits	Rejecting	Seizure
NCCEVN_ACCEPTING_CALL					X				X		
NCCEVN_ANSWERING_CALL	X				X				X		
NCCEVN_BILLING_INDICATION			X			X	X	X			
NCCEVN_BILLING_SET					X						
NCCEVN_CALL_CONNECTED		X					X	X			
NCCEVN_CALL_DISCONNECTED	X	X	X		X	X	X	X	X	X	X
NCCEVN_CALL_HELD (*)	X *	X *	X	X *	X *		X *	X *	X *	X *	X *

NCC event	NCC call state										
	Accepting	Answering	Connected	Disconnected	Incoming	Outbound initiated	Placing	Proceeding	Receiving digits	Rejecting	Seizure
NCCEVN_CALL_PROCEEDING							X				
NCCEVN_CALL_RELEASED				X		X					
NCCEVN_CALL_RETRIEVED (*)	X *	X *	X	X *	X *		X *	X *	X *	X *	X *
NCCEVN_CALL_STATUS_UPDATE	X	X	X	X	X	X	X	X	X	X	X
NCCEVN_CALLID_AVAILABLE	X	X	X	X	X	X	X	X	X	X	X
NCCEVN_EXTENDED_CALL_STATUS_UPDATE	X	X	X	X	X	X	X	X	X	X	X
NCCEVN_HOLD_REJECTED (*)	X *	X *	X	X *	X *		X *	X *	X *	X *	X *
NCCEVN_INCOMING_CALL									X		X
NCCEVN_PLACING_CALL						X					
NCCEVN_PROTOCOL_ERROR	X	X	X	X	X	X	X	X	X	X	X
NCCEVN_PROTOCOL_EVENT	X	X	X	X	X	X	X	X	X	X	X
NCCEVN_RECEIVED_DIGIT									X		X
NCCEVN_REJECTING_CALL	X				X				X		
NCCEVN_REMOTE_ALERTING			X					X			
NCCEVN_REMOTE_ANSWERED			X					X			
NCCEVN_RETRIEVE_REJECTED (*)	X *	X *	X	X *	X *		X *	X *	X *	X *	X *

Numerical event summary

The following table numerically lists the NCC events:

Hex	Decimal	Event name
0x001C2000	1843200	NCCEVN_PROTOCOL_ERROR

Hex	Decimal	Event name
0x001C2001	1843201	NCCEVN_ACCEPTING_CALL
0x001C2002	1843202	NCCEVN_ANSWERING_CALL
0x001C2003	1843203	NCCEVN_BILLING_INDICATION
0x001C2004	1843204	NCCEVN_BILLING_SET
0x001C2005	1843205	NCCEVN_CALLS_BLOCKED
0x001C2006	1843206	NCCEVN_CALLS_UNBLOCKED
0x001C2007	1843207	NCCEVN_CALL_CONNECTED
0x001C2008	1843208	NCCEVN_CALL_DISCONNECTED
0x001C2009	1843209	NCCEVN_CALL_HELD
0x001C200A	1843210	NCCEVN_PROTOCOL_EVENT
0x001C200B	1843211	NCCEVN_CALL_PROCEEDING
0x001C200C	1843212	NCCEVN_CALL_STATUS_UPDATE
0x001C200D	1843213	NCCEVN_HOLD_REJECTED
0x001C200E	1843214	NCCEVN_INCOMING_CALL
0x001C200F	1843215	NCCEVN_LINE_IN_SERVICE
0x001C2010	1843216	NCCEVN_LINE_OUT_OF_SERVICE
0x001C2011	1843217	NCCEVN_REMOTE_ALERTING
0x001C2012	1843218	NCCEVN_REMOTE_ANSWERED
0x001C2013	1843219	NCCEVN_PLACING_CALL
0x001C2014	1843220	NCCEVN_REJECTING_CALL
0x001C2015	1843221	NCCEVN_CALL_RELEASED
0x001C2016	1843222	NCCEVN_CALL_RETRIEVED
0x001C2017	1843223	NCCEVN_RETRIEVE_REJECTED

Hex	Decimal	Event name
0x001C2018	1843224	NCCEVN_SEIZURE_DETECTED
0x001C201B	1843227	NCCEVN_CAPABILITY_UPDATE
0x001C201C	1843228	NCCEVN_EXTENDED_CALL_STATUS_UPDATE
0x001C201D	1843229	NCCEVN_RECEIVED_DIGIT
0x001C2020	1843232	NCCEVN_BLOCK_FAILED
0x001C2021	1843233	NCCEVN_UNBLOCK_FAILED
0x001C2022	1843234	NCCEVN_CALLID_AVAILABLE
0x001C2119	1843481	NCCEVN_START_PROTOCOL_DONE
0x001C211A	1843482	NCCEVN_STOP_PROTOCOL_DONE

Using the event reference

A typical event description includes:

Description	A brief description of the meaning of the event.
State transition	If the event signifies a call or line state transition, the transition is described here. If the event is informational, None appears here.
Associated function	If the event is solicited (occurs as a result of a function call), the associated function or functions are listed here. If the event is unsolicited, Unsolicited appears here.
Fields	Lists information returned with the event, such as line or call handles or values.
Reason codes	If the event returns a value field, possible values are listed here.
Details	Lists any further information about the event, such as capabilitymask or bit settings that regulate whether or not the event occurs.

NCCEVN_ACCEPTING_CALL

The application accepted the call referenced by the call handle.

State transition

To accepting call state, from receiving digits or incoming call states.

Associated function

nccAcceptCall

Fields

Field	Description
callhd	Call handle.
value	The method by which the call was accepted.

Reason codes

Reason code	Description
NCC_ACCEPT_PLAY_RING	Play a ring.
NCC_ACCEPT_PLAY_SILENT	Play nothing.
NCC_ACCEPT_USER_AUDIO	Let user supply audio to play.

Details

NCCEVN_ACCEPTING_CALL indicates that the application's attempt to accept an incoming call (using **nccAcceptCall**) was successful. The call enters the accepting state. The application can answer or reject the call.

The event value field contains the method for accepting the call specified by the application when it invoked **nccAcceptCall**.

Call acceptance is not supported by all protocols. NCC_CAP_ACCEPT_CALL in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports the accepting call state.

NCCEVN_ANSWERING_CALL

The application answered the call referenced by the call handle.

State transition

To answering call state, from receiving digits, incoming, or accepting call states.

Associated function

nccAnswerCall

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_ANSWERING_CALL indicates that the application's attempt to answer an incoming call with **nccAnswerCall** was successful. The call enters the answering state.

NCCEVN_BILLING_INDICATION

A billing indication arrived.

State transition

None.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.
value	Billing units charged.

Details

NCCEVN_BILLING_INDICATION indicates that a billing indication was detected for an outbound call. This can be a billing pulse or some more complex indication (ISDN, SS7). The event value field may contain the billing units charged, depending upon the protocol implementation.

This event is generated only if the NCC_REPORT_BILLING bit is set in the NCC.START.eventmask parameter (see NCC API global parameters).

NCCEVN_BILLING_SET

Request to set the billing for an incoming call is complete.

State transition

None.

Associated function

nccSetBilling

Fields

Field	Description
callhd	Call handle.
value	Billing rate or reason code.

Reason codes

Reason code	Description
NCC_BILLINGSET_DEFAULT	Billing set to network default value.
NCC_BILLINGSET_FREE	Call is free.

Details

NCCEVN_BILLING_SET indicates that the billing information was successfully sent to the network. The event value field contains the billing rate or one of the reason codes listed previously.

Billing setting is not supported by all protocols. NCC_CAP_SET_BILLING in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports the ability to set the billing for a call.

NCCEVN_BLOCK_FAILED

Request to block calls on a line failed.

State transition

None.

Associated function

nccBlockCalls

Fields

Field	Description
ctahd	Line handle (context handle).
value	Reason for failure.

Reason codes

Reason code	Description
NCC_BLOCK_CAPABILITY_ERROR	The protocol does not support call blocking.
NCC_BLOCK_OUT_OF_SEQUENCE	The line is in a state that does not allow blocking.
NCC_BLOCK_TIMEOUT	Block command timed out.

Details

NCCEVN_BLOCK_FAILED indicates that a request to block calls on a line failed. This usually means that the function timed out before all existing calls on the line were released.

The event value field contains the reason why the request failed.

NCCEVN_CALL_CONNECTED

Both parties are now connected.

State transition

To connected state from answering, placing, or proceeding call states.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.
value	Indicates the reason why the call was connected.

Reason codes

Reason code	Description
NCC_CON_ANSWERED	Call answered by **nccAnswerCall**.
NCC_CON_CED	Outbound CED detected.
NCC_CON_DIALTONE_DETECTED	Outbound CP dialtone (may be disabled).
NCC_CON_PROCEEDING	Do not run call progress.
NCC_CON_RING_BEGIN	Outbound ring detected.
NCC_CON_RING_QUIT	Outbound ring stop (soft speaker).
NCC_CON_SIGNAL	Connect by out-of-band signal.
NCC_CON_SIT_DETECTED	Outbound SIT detected.
NCC_CON_TIMEOUT	Outbound timeout.
NCC_CON_VOICE_BEGIN	Outbound voice detected.
NCC_CON_VOICE_END	Outbound voice ended.
NCC_CON_VOICE_EXTENDED	Outbound voice extended.
NCC_CON_VOICE_LONG	Outbound voice long.
NCC_CON_VOICE_MEDIUM	Outbound voice medium.

Details

NCCEVN_CALL_CONNECTED indicates that a call reached the connected state. The event value field contains the reason for the connection.

NCCEVN_CALL_DISCONNECTED

(unsolicited) The remote party disconnected the referenced call.

(solicited) **nccDisconnectCall** was invoked. The call is now disconnected from the network.

State transition

To disconnected call state, from any other call state. May also indicate a line state change.

Associated function

nccDisconnectCall, unsolicited

Fields

Field	Description
callhd	Call handle.
value	Contains the reason why the disconnect occurred.

Reason codes

Reason code	Description
NCC_DIS_BUSY	Place call received busy tone.
NCC_DIS_CED	Place call received CED detected.
NCC_DIS_CLEARDOWN_TONE	In-band remote disconnect.
NCC_DIS_CONGESTION	Place call received reorder tone.
NCC_DIS_DIAL_FAILURE	Dial failed.
NCC_DIS_DIALTONE	Place call received dialtone after dial.
NCC_DIS_GLARE	Glare detected.
NCC_DIS_HOST_TIMEOUT	Host did not respond in time.
NCC_DIS_INCOMING_FAULT	A fatal fault occurred during the setup of an incoming call. This mapping is used for protocols that do not have a protocol-specific DIS_INCOMING_FAULT value.

Reason code	Description
NCC_DIS_NO_ACKNOWLEDGEMENT	Acknowledgement of the initiation of an outbound call was not detected.
NCC_DIS_NO_CS_RESOURCE	No CS resource to place call.
NCC_DIS_NO_DIALTONE	No dialtone to dialing out (LPS).
NCC_DIS_NO_LOOP_CURRENT	No loop current to dial out (LPS).
NCC_DIS_PROTOCOL_ERROR	Disconnected on protocol error.
NCC_DIS_REJECT_REQUESTED	Call disconnected by **nccRejectCall**.
NCC_DIS_REMOTE_ABANDONED	Loop start inbound stopped ringing.
NCC_DIS_REMOTE_NOANSWER	Place call received no answer.
NCC_DIS_RING_BEGIN	Place call received ring begin.
NCC_DIS_RING_QUIT	Place call received ringstop (soft speaker).
NCC_DIS_SIGNAL	Normal out-of-band remote disconnect.
NCC_DIS_UNASSIGNED_NUMBER	A signal or message was detected, meaning that the called address is not allocated.
NCC_DIS_SIGNAL_UNKNOWN	An unspecified disconnect signal or message was detected.
NCC_DIS_SIT_DETECTED	Place call received SIT tone.
NCC_DIS_TIMEOUT	Place call received timeout.
NCC_DIS_TRANSFER	Transfer completed.
NCC_DIS_VOICE_BEGIN	Place call received voice detected.
NCC_DIS_VOICE_END	Place call received voice ended.
NCC_DIS_VOICE_EXTENDED	Place call received voice extended.
NCC_DIS_VOICE_LONG	Place call received voice long.
NCC_DIS_VOICE_MEDIUM	Place call received voice medium.

Details

NCCEVN_CALL_DISCONNECTED indicates one of the following:

- The application has invoked **nccDisconnectCall**.
- The remote party has hung up. This can occur in almost any call state.
- The call is inbound and the application has rejected the call.
- (ADIMGR NCC implementation only) The call is outbound, the remote party has answered, and the call has failed to meet the criteria specified by the NCC_X.ADI_PLACECALL_PARMS connectmask passed with **nccPlaceCall**, or has met the criteria specified by the disconnectmask.
- A call is successfully transferred using **nccTransferCall** or **nccAutomaticTransfer** (see Transferring calls).

The event value field contains the reason that the call was disconnected.

A disconnected call is no longer considered active. If there are no active calls on a line (any calls on the line are either held or disconnected), the line state returns to idle.

Use **nccReleaseCall** to release disconnected calls.

If the NCC_CAP_DISCONNECT_IN_ANY_STATE indicator is set in the capabilitymask for the protocol, the application can invoke **nccDisconnectCall** to disconnect a call regardless of the call state. In this case, a call can reach disconnected call state from any state.

NCCEVN_CALL_HELD

(solicited) **nccHoldCall** or **nccAutomaticTransfer** was invoked; call is now held.

(unsolicited) The CPE placed the call on hold.

State transition

To idle line state, from active line state.

Associated function

nccHoldCall, **nccAutomaticTransfer**, unsolicited

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_CALL_HELD indicates that a call was placed on hold, either by the application or by a remote party. When a call is placed on hold, no call state change occurs. However, a call on hold is not considered active. If there are no active calls on a line (all calls on the line are either held or disconnected), the line state returns to idle.

Call holding is not supported by all protocols. The NCC_CAP_HOLD_CALL indicator in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports this event or not.

A call can only be placed on hold while in the connected state unless the NCC_CAP_HOLD_IN_ANY_STATE indicator is set in the capabilitymask for the protocol. In this case, the held call may change call states while on hold.

The application can invoke **nccGetCallStatus** to determine whether a call is on hold or not. If a call is on hold, the held field in the NCC_CALL_STATUS structure returned for that call contains a non-zero value.

NCCEVN_CALL_PROCEEDING

The switch accepted the call setup. The receiving side is being rung.

State transition

To proceeding call state, from placing call state.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_CALL_PROCEEDING indicates that the switch accepted the call setup request and is attempting to ring the receiving end. Call progress analysis is begun. The call state changes to proceeding.

NCCEVN_CALL_RELEASED

The application released the call and NCC released all internal resources for the call.

State transition

To undefined state, from disconnected or outbound initiated call states.

Associated function

nccReleaseCall

Fields

Field	Description
callhd	Call handle.
value	Reason for release.

Reason codes

Reason code	Description
NCC_RELEASED_GLARE	Outbound call was released because an incoming call was detected.
NCC_RELEASED_ERROR	Outbound call was released because an error occurred.

Details

NCCEVN_CALL_RELEASED is generated in response to **nccReleaseCall**. It indicates that the call handle is destroyed and the call is no longer in any state. If this call handle is currently referenced in the application, the application receives an error.

An outbound call may also be released if a glare (call collision) situation is detected while the call is in outbound initiated state. In this case, the event value field contains NCC_RELEASED_GLARE.

After the application receives this event, no more DSP-related events are generated for this call handle. Also, the application can no longer retrieve call status information for the call using **nccGetCallStatus** or **nccGetExtendedCallStatus**.

NCCEVN_CALL_RETRIEVED

(solicited) The application retrieved a call it placed on hold.

(unsolicited) A call was retrieved by this unsolicited event.

State transition

To active line state, from idle line state.

Associated function

nccRetrieveCall, **nccAutomaticTransfer**, unsolicited

Fields

Field	Description
callhd	Call handle.

Reason codes

Reason code	Description
NCC_DIS_***xxx***	Signifies that an automatic transfer failed. For a list of valid values, see NCCEVN_CALL_DISCONNECTED.

Details

NCCEVN_CALL_RETRIEVED indicates that a call on hold was retrieved either by the application or by a remote party. The call is now active. Since there is an active call on the line, the line state changes to active.

Call hold/retrieve is not supported by all protocols. NCC_CAP_HOLD_CALL in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports this capability or not.

The application can invoke **nccGetCallStatus** to determine whether a call is on hold or not. If a call is on hold, the held field in the NCC_CALL_STATUS structure returned for that call contains a non-zero value.

NCCEVN_CALL_STATUS_UPDATE

The information in the NCC_CALL_STATUS structure changed. The application can invoke **nccGetCallStatus** to get the updated information.

State transition

None.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.
value	Indicates the type of information that changed.

Reason codes

Reason code	Description
NCC_CALL_STATUS_CALLINGADDR	Calling address was updated (caller ID information).

Details

If call status information changes, the application receives NCCEVN_CALL_STATUS_UPDATE. The event value field indicates the type of information that changed. The application can then invoke **nccGetCallStatus** to get the updated information.

This event can occur in any call state, as long as the line state is active or idle.

This event is generated only if the NCC_REPORT_STATUSINFO bit is set in the NCC.START.eventmask parameter (see NCC API global parameters).

NCCEVN_CALLID_AVAILABLE

A call identifier (used to identify a call in a two channel transfer) is available.

State transition

None.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_CALLID_AVAILABLE indicates that a call identifier is available for two channel transfer. Upon receiving this event, the application can issue a two channel transfer. This event can occur in any call state.

This event is generated only if the NCC_REPORT_CALLID bit is set in the NCC.START.eventmask parameter (see NCC API global parameters). Default is that the event is not sent.

NCCEVN_CALLS_BLOCKED

The application blocked all calls on a line.

State transition

To blocking line state, from idle line state.

Associated function

nccBlockCalls

Fields

Field	Description
ctahd	Line handle (context handle).
value	Indicates the blocking method.

Reason codes

Reason code	Description
NCC_BLOCK_REJECTALL	Do not answer subsequent calls.
NCC_BLOCK_OUT_OF_SERVICE	Calls blocked by request of network.

Details

NCCEVN_CALLS_BLOCKED indicates that all calls are blocked on a line. It occurs as a response to an **nccBlockCalls** function invocation. The event value field indicates the blocking method specified by the application when it invoked **nccBlockCalls**.

The NCC API does not place the line in blocking line state until there are no calls on the line. Calls are not blocked until the NCCEVN_CALLS_BLOCKED event is received.

NCCEVN_CALLS_UNBLOCKED

The line is no longer blocked.

State transition

To idle line state, from blocking line state.

Associated function

nccUnblockCalls

Fields

Field	Description
ctahd	Line handle (context handle).

Details

NCCEVN_CALLS_UNBLOCKED indicates that a line is no longer in blocking line state. At this point, the application can receive or place calls on the line.

NCCEVN_CAPABILITY_UPDATE

Protocol capabilities changed.

State transition

None.

Associated function

Unsolicited

Fields

Field	Description
ctahd	Line handle (context handle).
value	Indicates the capability type that was changed.

Details

The application receives NCCEVN_CAPABILITY_UPDATE when the capabilities of a protocol change. The event value field indicates the capability that changed. The application can invoke **nccQueryCapability** to determine the current set of protocol capabilities.

NCCEVN_EXTENDED_CALL_STATUS_UPDATE

Call status made available with **nccGetExtendedCallStatus** has been updated.

State transition

None.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.
value	Indicates the kind of information that changed.

Reason codes

Protocol-specific.

Details

The application receives NCCEVN_EXTENDED_CALL_STATUS_ UPDATE when protocol-specific call status information changes. The event value field indicates the kind of information that changed. The application can then invoke **nccGetExtendedCallStatus** to retrieve this information.

This event is generated only if the NCC_REPORT_STATUSINFO bit is set in the NCC.START.eventmask parameter (see NCC API global parameters).

NCCEVN_HOLD_REJECTED

Request to put a call on hold was rejected.

State transition

None.

Associated function

nccHoldCall

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_HOLD_REJECTED indicates that the protocol rejected a request to put a call on hold. The call remains active.

Call hold/retrieve is not supported by all protocols. NCC_CAP_HOLD_CALL in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports call hold or not.

NCCEVN_INCOMING_CALL

All information related to an incoming call was collected and presented to the application.

State transition

To incoming call state, from seizure or receiving digits call states.

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_INCOMING_CALL indicates that NCC detected an incoming call and gathered all necessary call information (for example, digits). The call is now in the incoming state. At this point, the application decides whether to accept, answer, or reject the call.

NCCEVN_LINE_IN_SERVICE

The network placed a line in service.

State transition

To idle line state, from out of service line state.

Associated function

Unsolicited

Fields

Field	Description
ctahd	Line handle (context handle).

Details

NCCEVN_LINE_IN_SERVICE indicates that a line in out of service line state was placed in service. The line is now in an idle state.

NCCEVN_LINE_OUT_OF_SERVICE

The network placed a line out of service.

State transition

To out-of-service line state, from idle line state.

Associated function

Unsolicited

Fields

Field	Description
ctahd	Line handle (context handle).
value	Reason for placing the line out of service.

Reason codes

Reason code	Description
NCC_OUT_OF_SERVICE_DIGIT_TIMEOUT	DID, timed out waiting for digits.
NCC_OUT_OF_SERVICE_NO_LOOP_CURRENT	No loop current to place calls.

Reason code	Description
NCC_OUT_OF_SERVICE_PERM_SIGNAL	Timed out waiting for remote end.
NCC_OUT_OF_SERVICE_REMOTE_BLOCK	Remote end is blocking calls.
NCC_OUT_OF_SERVICE_WINK_STUCK	OGT, wink is too long.
NCC_OUT_OF_SERVICE_NO_DIGITS	No digits received after seizure.
NCC_OUT_OF_SERVICE_LINE_FAULT	Line in inappropriate state.

Details

NCCEVN_LINE_OUT_OF_SERVICE indicates that a line was taken out of service. The event value field indicates the reason for taking the line out of service.

NCCEVN_PLACING_CALL

A call was successfully placed.

State transition

To placing call state, from outbound initiated call state.

Associated function

nccPlaceCall

Fields

Field	Description
callhd	New call handle.

Details

NCCEVN_PLACING_CALL indicates that a call placement attempt made by the application was successful. NCC seized the line, resolved glare, and created a call handle. The call handle can now be used for all call operations.

NCCEVN_PROTOCOL_ERROR

A two channel call transfer failed or an error condition occurred. The call may be in an unusable state.

State transition

None.

Associated function

Unsolicited, **nccTransferCall**

Fields

Field	Description
callhd	New call handle.
value	Qualifies the event.
size	Protocol-specific reason code for two channel transfer.

Reason codes

Reason code	Description
NCC_PROTERR_BAD_CALLERID	Caller ID information was incorrect.
NCC_PROTERR_CAPABILITY_ERROR	Generic: last command is unsupported.
NCC_PROTERR_COMMAND_OUT_OF_SEQUENCE	Command made when call/line was in an inappropriate state.
NCC_PROTERR_DIGIT_TIMEOUT	Timeout in digit reception.
NCC_PROTERR_EVENT_OUT_OF_SEQUENCE	Event received when call/line was in an inappropriate state.
NCC_PROTERR_EXTRA_DIGITS	DID: more digits received than expected.
NCC_PROTERR_FALSE_SEIZURE	Event received when seizure has been aborted.
NCC_PROTERR_INVALID_DIGIT	Invalid compelled digit received.
NCC_PROTERR_NO_CS_RESOURCE	Call setup resource unavailable.
NCC_PROTERR_PREMATURE_ANSWER	OGT: answer before all digits given.
NCC_PROTERR_TCT_FAILED	Two channel call transfer failed.
NCC_PROTERR_TIMEOUT	A timeout occurred.

Details

Consult the protocol documentation to learn how to handle NCCEVN_PROTOCOL_ERROR.

NCCEVN_PROTOCOL_EVENT

(solicited) The application invoked a protocol-specific function on a call (**nccSendCallMessage** or **nccSendLineMessage**), and that function was executed.

(unsolicited) The protocol implementation generated an unsolicited event.

State transition

None.

Associated function

nccSendCallMessage, **nccSendLineMessage**, unsolicited

Fields

Field	Description
callhd	Call handle.
value	Qualifies the event for the specific protocol implementation.

Reason codes

Reason code	Description
NCC_DBG_***xxx***	Protocol-specific reason code.

Details

Consult the protocol documentation to learn how to handle NCCEVN_PROTOCOL_EVENT.

NCCEVN_RECEIVED_DIGIT

Digits are being received in an overlapped fashion for an incoming call.

State transition

From seizure call state, to receiving digits call state (if the call is not already in this state).

Associated function

Unsolicited

Fields

Field	Description
callhd	Call handle.
value	A single digit received from the network.

Details

NCCEVN_RECEIVED_DIGIT indicates that a call address or subaddress digit was received from the network. The application may receive several of these events, as digits are received in an overlapped fashion for an incoming call. The value field contains the digit.

When the first digit is received, the call state changes from seizure to receiving digits. The call remains in this state until NCCEVN_INCOMING_CALL is received.

This event is generated only if the NCC.START.overlappedreceiving bit is set (see NCC API global parameters).

NCCEVN_REJECTING_CALL

(solicited) **nccRejectCall** was invoked. The call is being rejected.

(unsolicited) The application failed to answer, accept, or reject the call in a timely fashion. The call is automatically rejected.

State transition

To rejecting call state, from incoming, receiving digits, or accepting call states.

Associated function

nccRejectCall, unsolicited

Fields

Field	Description
callhd	Call handle.
value	Reason the call was rejected.

Reason codes

Reason code	Description
NCC_REJECT_HOST_TIMEOUT	Reject inbound with default tone because PC did not respond in time.
NCC_REJECT_PLAY_BUSY	Reject inbound with busy (if able).
NCC_REJECT_PLAY_REORDER	Reject inbound with reorder (if able).
NCC_REJECT_PLAY_RINGTONE	Reject inbound with ringtone (if able).
NCC_REJECT_USER_AUDIO	Reject inbound with user audio (if able).

Details

The application receives NCCEVN_REJECTING_CALL in either of the following situations:

- The application invoked **nccRejectCall** to reject an incoming call.
- The application failed to accept, answer, or reject an incoming call within the period of time specified by the NCC.START.waitforPCtime parameter.

The call enters the rejecting call state.

The event value field contains the reason why the call was rejected.

NCCEVN_REMOTE_ALERTING

The remote end is ringing.

State transition

None.

Associated function

nccPlaceCall

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_REMOTE_ALERTING indicates that the remote end is in an alerting state (is ringing).

This event is generated only if the NCC_REPORT_ALERTING bit is set in the NCC.START.eventmask parameter (see NCC API global parameters).

NCCEVN_REMOTE_ANSWERED

The remote end is answering the call.

State transition

None.

Associated function

nccPlaceCall

Fields

Field	Description
callhd	Call handle.
value	Reason this event took place.

Reason codes

Reason code	Description
NCC_ANSWER_MODEM	Due to modem detection.
NCC_ANSWER_SIGNAL	Due to out-of-band signaling.
NCC_ANSWER_VOICE	Due to voice detection.

Details

NCCEVN_REMOTE_ANSWERED is generated at the first positive indication that the remote party has answered (out-of-band signaling, voice, or modem tone). The event value field contains the indication type.

This event is generated only if the NCC_REPORT_ANSWERED bit is set in the NCC.START.eventmask parameter (see NCC API global parameters).

NCCEVN_RETRIEVE_REJECTED

A retrieve call request was rejected.

State transition

None.

Associated function

nccRetrieveCall

Fields

Field	Description
callhd	Call handle.

Details

NCCEVN_RETRIEVE_REJECTED indicates that the application attempted to retrieve a call it placed on hold, but the network did not allow it.

Call hold/retrieve is not supported by all protocols. NCC_CAP_RETRIEVE_CALL in the capabilitymask returned by **nccQueryCapability** indicates if the current protocol supports call hold or not.

NCCEVN_SEIZURE_DETECTED

The network seized the line.

State transition

To seizure call state.

Associated function

Unsolicited

Fields

Field	Description
callhd	New call handle.

Details

NCCEVN_SEIZURE_DETECTED indicates that NCC detected an incoming call. A call handle is created for the call. The call handle can now be used for all subsequent operations for this call.

The new call begins in the seizure call state.

NCCEVN_START_PROTOCOL_DONE

The application attempted to start a protocol on a context (line handle).

State transition

(If successful) to idle line state, from uninitialized state.

Associated function

nccStartProtocol

Fields

Field	Description
ctahd	Line handle.

Reason codes

Reason code	Description
CTA_REASON_FINISHED	Protocol started successfully. The line state changes from uninitialized to idle.
NCCREASON_OUT_OF_RESOURCES	Protocol was not started because the mediamask parameter was not set correctly. The line remains in an uninitialized state. (See your hardware documentation for details.)
NCCREASON_WRONG_CC_MODE	Protocol was not started because an invalid call control function (such as **adiStartProtocol**) was invoked to start the protocol. Use **nccStartProtocol** to start a protocol on a context if the NCC API has been opened on that context.

Details

NCCEVN_START_PROTOCOL_DONE acknowledges a protocol startup attempt. The event value field indicates whether the attempt was successful or not.

The event value field can also contain the error code CTAERR_BOARD_ERROR, indicating that an error occurred on the board when the application attempted to start a protocol. If you configured the echo canceller, verify that the DSP file was downloaded to the board.

NCCEVN_STOP_PROTOCOL_DONE

The protocol stopped running on the context. The line is uninitialized.

State transition

To uninitialized line state, from any other state.

Associated function

nccStopProtocol

Fields

Field	Description
ctahd	Line handle (context handle).

Reason codes

Reason code	Description
CTA_REASON_FINISHED	Protocol stopped successfully.

Details

NCCEVN_STOP_PROTOCOL_DONE indicates that a protocol successfully stopped on a context (line handle). The line state changes to uninitialized. The event value field contains CTA_REASON_FINISHED.

NCCEVN_UNBLOCK_FAILED

Request to unblock calls on a line failed.

State transition

None.

Associated function

nccUnblockCalls

Fields

Field	Description
ctahd	Line handle (context handle).
value	Reason for failure.

Reason codes

Reason code	Description
NCC_UNBLOCK_CAPABILITY_ERROR	The protocol does not support call unblocking.
NCC_UNBLOCK_OUT_OF_SEQUENCE	The line is in a state that does not allow unblocking.
NCC_UNBLOCK_TIMEOUT	Unblock command timed out.

Details

NCCEVN_UNBLOCK_FAILED indicates that a request to unblock calls on a line failed. The event value field contains the reason the request failed.

12. Demonstration program

Before running nccxfer

NCC provides the *nccxfer* program to demonstrate transferring an incoming call. The software includes the *nccxfer* executable program and the program's source and makefiles.

Before you start the demonstration program, ensure that:

- NaturalAccess is properly installed.
- The AG Series or CG Series board is executing.

Refer to the appropriate installation manual for your operating system for details on installation.

Running nccxfer

nccxfer demonstrates rerouting an incoming call.

Usage

```
nccxfer [options]
```

where ***options*** are:

Option	Description
-A ***xxxmgr***	Specifies the NaturalAccess manager to use for the ADI API. Default is ADIMGR.
-b ***n***	Specifies the board number ***n***. Default is 0.
-s ***n***:***m***	Specifies the MVIP stream and timeslot for the first channel. Default is 0:0.
-p ***protocol***	Specifies the protocol to run. Default is lps0.
-i ***n***	Specifies the number of call iterations (0=infinite).
-m ***n***	Specifies the method of transfer. Valid values for ***n***:

Value	Description
1	Transfer on PROCEEDING (after dial) (default)
2	Transfer on ALERTING (ringing)
3	Transfer on CONNECTED (first voice or signal)
4	Manual (call screening)

Featured functions

nccAutomaticTransfer, **nccTransferCall**

Description

Party A calls party B (which is the demonstration program) and requests a transfer to party C. *nccxfer* (party B) then calls party C. It reports success or failure at one of several stages, depending on the selected transfer method. If -m 4 is specified on the command line, *nccxfer* asks party C whether it should complete the transfer, or report failure.

Procedure

Begin with three telephone lines: two (A and C) with two 2500-type telephone sets, and one (B) connected to the board. Lines B and C must be connected to a common PBX.

Complete the following steps to run *nccxfer*:

Step	Action
1	Specify the proper board and MVIP stream, and start the configuration utility.
2	Start *nccxfer* by entering the following at the prompt: `nccxfer [ -b n -s n:m -p protocol -i n -m n ]` Make sure to specify a transfer method.
3	From line A, call line B. The demonstration answers.
4	When the program requests an extension to try, dial the extension of line C. The demonstration calls line C.
5	Answer line C.

With transfer method 4, the demonstration offers you a chance to refuse the call. The other transfer modes report success or failure at earlier stages, sometimes before the transfer is actually completed.

13. NCC API errors

Overview of the NCC API errors

All NaturalAccess functions return SUCCESS (0) or an error code indicating that the function failed and the reason for the failure.

NaturalAccess error codes are defined in the *ctaerr.h* include file. The error codes are prefixed with CTAERR_. NCC API error codes are defined in the *nccerr.h* include file. The error codes are prefixed with NCCERR_.

Alphabetical error summary

The following table alphabetically lists errors that can occur as a direct result of an NCC API function call. All errors are 32 bits.

Error name	Hex	Decimal	Description
CTAERR_BAD_ARGUMENT	0x7	7	A function argument had an invalid value, or a required pointer argument was NULL. Check all arguments for valid types and ranges.
CTAERR_BAD_SIZE	0xB	11	A ***size*** argument was too small to receive a data structure, or a play or record buffer was not a multiple of the frame size for the specified encoding. Check all arguments for valid sizes.
CTAERR_BOARD_ERROR	0x3	3	An error occurred on the board when the application attempted to start a protocol. If you configured the echo canceller, verify that the DSP file was downloaded to the board.
CTAERR_INVALID_CTAHD	0x5	5	An invalid context handle was passed as an argument to a function, or the context was destroyed by another thread.

Error name	Hex	Decimal	Description
CTAERR_INVALID_HANDLE	0x20	32	An invalid handle was passed as an argument to this function.
CTAERR_INVALID_STATE	0xC	12	This function is not valid in the current line state.
CTAERR_SVR_COMM	0x41	65	Server communication error.
NCCERR_ADDRESS_BLOCKED	0x001C0003	1835011	Too many calls placed to this address, or a call was placed too recently.
NCCERR_INVALID_CALL_STATE	0x001C0002	1835010	This function is not valid in the current call state.
NCCERR_NO_CALLID	0x001C0004	1835012	Call handles are not on the same line (supervised transfer), or no call identifier exists (two channel transfer).
NCCERR_NOT_CAPABLE	0x001C0001	1835009	A function or parameter is not supported by the current protocol. **nccQueryCapability** can be used to determine features supported by the protocol.

Numerical error summary

The following table numerically lists errors that can occur as a direct result of an NCC API function call:

Hex	Decimal	Error name
0x3	3	CTAERR_BOARD_ERROR
0x5	5	CTAERR_INVALID_CTAHD
0x7	7	CTAERR_BAD_ARGUMENT
0xB	11	CTAERR_BAD_SIZE
0xC	12	CTAERR_INVALID_STATE
0x20	32	CTAERR_INVALID_HANDLE

Hex	Decimal	Error name
0x001C0001	1835009	NCCERR_NOT_CAPABLE
0x001C0002	1835010	NCCERR_INVALID_CALL_STATE
0x001C0003	1835011	NCCERR_ADDRESS_BLOCKED
0x001C0004	1835012	NCCERR_NO_CALLID
0x00000041	65	CTAERR_SVR_COMM

14. NCC API parameters

Overview of the NCC API parameters

The behavior of most NCC functions, and of the operating protocol itself, can be controlled using parameters. There are three types of NCC parameters:

NCC parameter type	Description	Documented in...
Implementation-independent, protocol-independent	These parameters determine basic NCC API functionality.	This section.
Implementation-dependent, protocol-independent	These parameters determine the overall behavior of a family of protocols.	Your protocol-specific documentation.
Implementation-dependent, protocol-dependent	Various sets of these parameters control the behavior of specific protocols	Your protocol-specific documentation.

Parameter files

NCC parameters are stored in binary parameter files. ASCII versions of these files are also available. These parameter files have country- or protocol-specific values that are used for the target country for your application. NCC API-related parameter files use the following naming conventions:

Name	Contents
nccstart.pf	Implementation-independent, protocol-independent NCC parameters. Since these go into effect when a protocol is started with **nccStartProtocol**, the word start appears in the name.
*nccx**imp**.pf*	Implementation-dependent, protocol-independent NCC parameters. ***imp*** represents the implementation. For example, for the ADI implementation, the filename is *nccxadi.pf*.
*nccx**prt**.pf*	Implementation-dependent, protocol-dependent NCC parameters. ***prt*** represents the protocol or protocol family that the file is for. For example, the CAS API (channel-associated signaling) protocols have filenames such as *nccxcas.pf* and *nccxlps.pf*.

Parameter names

NCC API parameters are named according to the following syntax:

```
SvcName.[X.][implementation_]category.[subStructure.]fieldName
```

where:

Name	Description
SvcName	The API (service) to which the parameter belongs. For NCC API parameters, this is NCC.
X	(optional) This indicator appears in the parameter name unless the parameter is supported by all implementations.
implementation	(optional) The implementation that the parameter is for. Empty if the parameter is for all implementations.
category	The parameter category (usually represents the function or protocol that the parameter concerns).
subStructure	(optional) Allows multiple, logically related ***fieldName***s to be nested under a category.
fieldName	The actual parameter.

The following table lists sample parameter names:

Parameter Name	Description
NCC.START.***xxx***	NCC API implementation-independent, protocol-independent parameters. These go into effect when **nccStartProtocol** is invoked.
NCC.X.ADI_START.***xxx***	Protocol-independent parameters for the ADI implementation of the NCC API. These go into effect when **nccStartProtocol** is invoked. For field descriptions, see NCC.X.ADI_START.
NCC.X.ADI_PLACECALL.***xxx***	Protocol-independent parameters for the ADI implementation of the NCC API. These go into effect when **nccPlaceCall** is invoked. For field descriptions, see NCC.X.ADI_PLACECALL.
NCC.X.ADI_ISDN.ACCEPTCALL.***xxx***	ISDN Software parameters for the ADI implementation of the NCC API. These go into effect when **nccAcceptCall** is invoked.

Loading and changing parameters

A parameter file must be in one of the directories specified with the AGLOAD environment variable for NaturalAccess to load it. When the function associated with the parameter type is invoked, the parameters go into effect.

Each parameter structure has a set of default values that is sufficient for many configurations. The parameters can, however, be modified to:

- Enable or disable function features.
- Adapt the function for unusual configurations.

Note: Values for certain protocol-dependent parameters should not be changed. Changing their values may affect the regulatory approvals in the target country. For more information, refer to the protocol-specific documentation.

Complete the following steps to change parameter values in a *.pf* file:

Step	Action
1	Modify the value in the corresponding *.par* file.
2	Parse the *.par* file.
3	Do one of the following tasks: • Invoke **ctaSetParmByName** for each parameter specified in the file, to set a new default value. (For an example of this, see the **DemoLoadParameters** function in the demonstration library supplied with NaturalAccess.) • Use the *ctdaemon* program to set the parameters system-wide. See the *Dialogic® NaturalAccess™ Software Developer's Manual* for more information. • Invoke **ctaLoadParameterFile** from within the application.

Parameter modification must take place before **nccStartProtocol** is invoked to start the protocol (as described in Starting a protocol on a context). When the function is invoked, the TCP is programmed as specified by the parameters.

Modify parameters before you invoke **nccStartProtocol** to start the protocol (as described in Starting a protocol on a context). When you invoke the function, the protocol is programmed as specified by the parameters.

NCC API global parameters

This topic describes the following NCC implementation-independent, protocol-independent parameters, defined in *nccstart.pf*:

- NCC.START structure
- NCC.START.callproggenerate parameters
- NCC.START.eventmask bits

The parameters are alphabetized by category and subcategories. Within each category, fields are listed alphabetically. For information on implementation-dependent or protocol-dependent parameters, see your protocol-specific documentation.

NCC.START structure

NCC.START				
Dependent Function(s): nccStartProtocol				
Field name	**Type**	**Default**	**Units**	**Description**
eventmask	DWORD	0x0000	mask	Determines which events are sent to the application.
debugflag	WORD	0	integer	Reports low-level debug events to the application. Off (0) or on (non-zero).
waitforPCtime	WORD	10000	ms	Specifies the time to wait for the application to respond after an NCCEVN_INCOMING_CALL. If a response is not received within the timeout, the incoming call is rejected.
overlappedreceiving	WORD	0	integer	Determines if the protocol will receive digits in overlapped receiving mode.

NCC.START.callproggenerate parameters

NCC.START.callproggenerate				
Dependent function(s): nccStartProtocol				
Field name	**Type**	**Default**	**Units**	**Description**
dialtonefreq1	WORD	350	Hz	First frequency of dialtone.
dialtonefreq2	WORD	440	Hz	Second frequency of dialtone.
dialtoneontime1	WORD	-1	ms	Cadenced dialtone: first ON time.
dialtoneontime2	WORD	0	ms	Cadenced dialtone: second ON time.
dialtoneofftime1	WORD	0	ms	Cadenced dialtone: first OFF time.
dialtoneofftime2	WORD	0	ms	Cadenced dialtone: second OFF time.
dialtonelevel	WORD	150	IDU	Dialtone amplitude.
ringfreq1	WORD	440	Hz	First frequency of ringback tone.
ringfreq2	WORD	480	Hz	Second frequency of ringback tone.

NCC.START.callproggenerate				
ringontime	WORD	1000	ms	Ringback tone ON time.
ringofftime1	WORD	3000	ms	Ringback tone first (or only) OFF time.
ringofftime2	WORD	0	ms	Ringback tone second OFF time.
ringtonelevel	WORD	112	IDU	Ringback tone amplitude.
busyfreq1	WORD	480	Hz	First frequency of busy tone.
busyfreq2	WORD	620	Hz	Second frequency of busy tone.
busyontime	WORD	500	ms	Busy tone ON time.
busyofftime	WORD	500	ms	Busy tone OFF time.
busytonelevel	WORD	63	Hz	Busy tone amplitude.
fastbusyfreq1	WORD	480	Hz	First frequency of fast busy (reorder) tone.
fastbusyfreq2	WORD	620	Hz	Second frequency of fast busy (reorder) tone.
fastbusyontime	WORD	250	ms	Fast busy tone ON time.
fastbusyofftime	WORD	250	ms	Fast busy tone OFF time.
fastbusytonelevel	WORD	63	IDU	Fast busy tone amplitude.

NCC.START.eventmask bits

The NCC.START.eventmask parameter controls which events are returned to the application. The following table lists the eventmask bits and the events they control:

Value	Bit name	Event reported
0x4	NCC_REPORT_ALERTING	NCCEVN_REMOTE_ALERTING
0x8	NCC_REPORT_ANSWERED	NCCEVN_REMOTE_ANSWERED
0x20	NCC_REPORT_BILLING	NCCEVN_BILLING_INDICATION
0x40	NCC_REPORT_STATUSINFO	NCCEVN_CALL_STATUS_UPDATE NCCEVN_EXTENDED_CALL_STATUS_UPDATE
0x10000	NCC_REPORT_CALLID	NCCEVN_CALLID_AVAILABLE

NCC.X.ADI_PLACECALL

Dependent function: **nccPlaceCall**

Note: All Field names in this structure are prefaced with NCC.X.ADI_PLACECALL; for example, NCC.X.ADI_PLACECALL.mediamask.

Field name	Type	Default	Units	Description
connectmask	DWORD	0x0103	mask	Determines how an outbound call gets connected. Valid values are: • 0x0001: Connect on out of band signal. • 0x0002: Connect on call progress (cp) beginning of voice. • 0x0004: Connect on call progress medium length voice detected. • 0x0008: Connect on call progress long voice. • 0x0010: Connect on call progress extended voice. • 0x0020: Connect on call progress voice end. • 0x0080: Connect on call progress ring stopping. • 0x0100: Connect on call progress CED detected. • 0x0200: Connect on call progress dial tone detected. • 0x0400: Connect on call progress SIT tone detected. • 0x0800: Connect on call progress ring begin. • 0x8000: Do not run call progress.

Field name	Type	Default	Units	Description
disconnectmask	DWORD	0x0040	mask	Determines how to abandon an outgoing call. Valid values are: • 0x0002: Disconnect on call progress (cp) beginning of voice. • 0x0004: Disconnect on call progress medium voice detected. • 0x0008: Disconnect on call progress long voice detected. • 0x0010: Disconnect on call progress extended voice detected. • 0x0020: Disconnect on call progress voice ended. • 0x0040: Disconnect on call progress timeout. • 0x0080: Disconnect on call progress ring tone stopped. • 0x0100: Disconnect on call progress CED tone detected. • 0x0800: Disconnect on call progress ring begin.
callprog.timeout	DWORD	1000	ms	Maximum time that can elapse with no stimulus from the network before call progress stops with a reason of timeout. Valid range is 1 through 65535. If the value is set to zero, the timer is disabled.
callprog.busycount	DWORD	4	count	Number of non-precise busy tones that must occur before busy or fast busy is reported. Valid range is 1 through 32767.
callprog.ringcount	DWORD	7	count	Number of ring tones that must occur before NO_ANSWER is reported. Valid range is 1 through 32767.
callprog.maxreorder	DWORD	700	ms	Threshold time defining the total time period (on time plus off time) for distinguishing between fast busy (reorder) and slow busy. Valid range is 0 through 32767.

Field name	Type	Default	Units	Description
callprog.maxbusy	DWORD	1500	ms	Threshold time defining the total time period (on time plus off time) for distinguishing between slow busy and ringing tone. Valid range is 0 through 32767.
callprog.maxring	DWORD	3000	ms	Maximum duration of a tone to distinguish a ringing tone from a dial tone. Valid range is 0 through 32767.
callprog.maxringperiod	DWORD	8000	ms	Length of time of the last ringing tone plus the silence that follows, before call progress reports a ringing-ended event.
callprog.voicemedium	DWORD	3000	ms	Minimum length of time voice must be detected before call progress reports a medium-voice event.
callprog.voicelong	DWORD	6000	ms	Minimum length of time voice must be detected before call progress reports a long-voice event.
callprog.voicextended	DWORD	9000	ms	Minimum length of time voice must be detected before call progress reports an extended-voice event.
callprog.silencetime	DWORD	1500	ms	Minimum length of a silent period after voice is detected before call progress reports a voice-ended event.
callprog.precqualtime	DWORD	150	ms	Precise tone qualification time. All precise tones must be longer than this time to qualify.

Field name	Type	Default	Units	Description
callprog.precmask	DWORD	7	mask	Mask to control which precise detectors to run. Form a value by using the OR operator with any of the following bit masks: • NCC_CPMSK_PRECISE_CED (0x0001): CED tone modem • NCC_CPMSK_PRECISE_SIT (0x0002): SIT • NCC_CPMSK_PRECISE_BUSY (0x0004): Busy and reorder tone (US) • NCC_CPMSK_PRECISE_425 (0x0008) 425 Hz tone (busy and reorder tone, non-US) • NCC_CPMSK_PRECISE_SITEXT (0x0010): SIT type reporting • NCC_CPMSK_PRECISE_TDD (0x0020): TDD/TTY device • NCC_CPMSK_PRECISE_NU (0x0040): Unassigned number Only three of the four detectors can run concurrently. If all four detectors are specified, busy and reorder tones are determined by cadence alone. The SIT, CED, and TDD/TTY detectors are enabled. Busy and reorder tone (bit value 0x0004) and the 425 Hz tone selection (bit value 0x0008) are mutually exclusive. If you choose both, only the 425 Hz filter takes effect.
callprog.stopmask	DWORD	0	mask	Mask to control which events cause call progress to stop. A value can be formed by using the OR operation with any of the following values: • 0x0001: Ring tone • 0x0002: Ring end • 0x0004: Voice begin • 0x0008: Medium voice duration • 0x0010: Long voice duration • 0x0020: Extended voice duration • 0x0040: Voice end

Field name	Type	Default	Units	Description
callprog.silencelevel	INT32	-40	dBm	Maximum signal level that is considered to be silence. Valid board range is -46 through -34.
callprog.voicetoneratio	DWORD	0x30000	IDU	Do not modify.
callprog.qualtonetime1	DWORD	60	ms	Do not modify.
callprog.qualtonetime2	DWORD	80	ms	Do not modify.
callprog.qualvoicetime1	DWORD	60	ms	Do not modify.
callprog.qualvoicetime2	DWORD	60	ms	Do not modify.
callprog.leakagetime	DWORD	8	ms	Do not modify.
callprog.noiselevel	DWORD	0x14000	IDU	Do not modify.

NCC.X.ADI_START

Dependent function: nccStartProtocol

Note: All field names in this structure are prefaced with NCC.X.ADI_START; for example, NCC.X.ADI_START.mediamask.

Field name	Type	Default	Units	Description
mediamask	DWORD	0x001F	mask	Controls which functions are running or reserved when the call enters the connected (conversation) state. (The NOCC protocol enters this state immediately). Reserved indicates that the DSP MIPS are committed to the operation before the operation actually starts. The application must reserve DSP resources in advance by using this parameter for DTMF detection, silence detection, cleardown detection, and echo cancelation. See callctl.mediamask valid values.

Field name	Type	Default	Units	Description
debugmask	DWORD	0x0000	mask	Not applicable.
protocoldebugmask	DWORD	0x0000	mask	Not used.
dial.method	DWORD	0	mask	Not applicable.
dial.breaktime	DWORD	60	ms	Not applicable.
dial.maketime	DWORD	40	ms	Not applicable.
dial.interpulse	DWORD	700	ms	Not applicable.
dial.flashtime	DWORD	500	ms	Not applicable.
dial.shortpause	DWORD	2000	ms	Not applicable.
dial.longpause	DWORD	5000	ms	Not applicable.
dial.dtmfampl1	INT32	-6	dBm	Not applicable.
dial.dtmfampl2	INT32	-4	dBm	Not applicable.
dial.dtmfontime	DWORD	80	ms	Not applicable.
dial.dtmfofftime	DWORD	80	ms	Not applicable.
dial.dialtonewait	DWORD	5000	ms	Not applicable.
dial.tonefreq1	DWORD	350	Hz	Not applicable.
dial.tonebandw1	DWORD	40	Hz	Not applicable.
dial.tonefreq2	DWORD	440	Hz	Not applicable.
tonebandw2	DWORD	40	Hz	Not applicable.
dial.tonequalampl	INT32	-28	dBm	Not applicable.
dial.tonequaltime	DWORD	50	ms	Not applicable.
dial.tonereflevel	DWORD	0xB000	IDU	Not applicable.
dial.reserved	DWORD	0	internal	Not applicable.

Field name	Type	Default	Units	Description
dial.tonetotaltime	DWORD	0	ms	Total time for which the interrupted dialtone must be present. Valid values are: 0: Does not detect an interrupted dialtone. ***n*** (where ***n*** is 1 or higher): Detects an interrupted dialtone that is present for at least ***n*** ms.
dtmfdet.columnfour	DWORD	1		Flag that indicates whether to detect the A, B, C, and D DTMF digits. Set this value to 1 to detect these digits, or 0 to ignore them.
dtmfdet.onqualampl	INT32	-39	dBm	Minimum signal level recognized as a DTMF signal. Valid range is -51 through -15.
dtmfdet.onthreshold	DWORD	0xCAB0	IDU	Do not modify.
dtmfdet.onqualtime	DWORD	50	ms	Minimum duration of a recognized DTMF signal before a digit event is emitted. Valid range is 22 through 32767.
dtmfdet.offqualampl	INT32	-45	dBm	Minimum signal required to maintain recognition of a DTMF signal once recognition starts. Valid range is -51 through 15.
dtmfdet.offthreshold	DWORD	0x92E0	IDU	Do not modify.
dtmfdet.offqualtime	DWORD	40	ms	Minimum duration of absence of a recognized DTMF signal before an end-of-digit event is emitted. Valid range is 5 through 32767.

Field name	Type	Default	Units	Description
echocancel.mode	DWORD	0	Bit field	Controls echo canceler operation. Set the mode to one of the following: 0 = No echo cancelation. 1 = Use internal defaults for filter length and adaptation time based on board type. 2 = Use specified values. 3 = Ignore specified filterlength and adapttime values for adiModifyEchoCanceller only. Additional values can be formed by using the OR operation. See echocancel.mode valid values.
echocancel.filterlength	DWORD	0	ms	Filter length of echo canceler for echocancel.mode = 2. Set this value to 0 to omit echo canceling. Valid range is 0 through 20. Higher values require more DSP processing power.
echocancel.adapttime	DWORD	0	ms	Echo canceler adaptation time for echocancel.mode = 2 . Valid range is 100 through 1000. Lower values require more DSP processing power.
echocancel.gain	INT32	0	dB	Amount of amplification applied to echo-canceled output. Valid range is -54 through 24.
echocancel.predelay	DWORD	0	ms	Output sample delay. Valid range is 0 through 9. Valid range is 0 through 20.
cleardown.freq1	DWORD	350	Hz	Not applicable.
cleardown.bandw1	DWORD	40	Hz	Not applicable.
ceardown.freq2	DWORD	440	Hz	Not applicable.

Field name	Type	Default	Units	Description
cleardown.bandw2	DWORD	40	Hz	Not applicable.
cleardown.qualampl	INT32	-28	dBm	Not applicable.
cleardown.qualtime	DWORD	1000	ms	Not applicable.
cleardown.reflevel	DWORD	0xB000	IDU	Not applicable.
cleardown.reserved	DWORD	0	internal	Not applicable.
cleardown.tonecount	DWORD	0	integer	Not applicable.
cleardown.minontime	DWORD	0	ms	Not applicable.
cleardown.maxontime	DWORD	0	ms	Not applicable.
cleardown.minofftime	DWORD	0	ms	Not applicable.
cleardown.maxofftime	DWORD	0	ms	Not applicable.

callctl.mediamask valid values

A value can be formed by using the OR operation with any of the following values:

Value	Description
0x0001	Reserve DTMF detector.
0x0002	Reserve silence detector.
0x0004	Reserve cleardown detector.
0x0008	Start DTMF detector.
0x0010	Start echo canceler.

echocancel.mode valid values

Value	Description
0x0004	Not applicable.
0x0008	Enable echo suppressor.
0x0010	Do not reset echo canceler.

Value	Description
0x0020	Disable filter taps adaptation.
0x0040	Bypass echo cancelation.
0x0080	Request status of echo canceler.
0x0100	Enable auto-status event generation when status of echo canceler changes.
0x0200	Enable comfort noise generation.

15. Index