Dialogic
Making Innovation Thrive™

Dialogic® NaturalAccess™ ISDN Software Developer's Manual

October 2010 64-0508-03 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
9000-6407-20	September, 2000	SJC, CT Access 4.0
9000-6407-21	March, 2001	SJC, NACD 2000-2
9000-6407-22	April, 2001	CYF, NACD 2001-1 Beta
9000-6407-23	August, 2001	SJC, NACD 2001-1
9000-6407-24	November, 2001	SJC, NACD 2002-1 Beta
9000-6407-25	May, 2002	LBG, NACD 2002-1
9000-6407-26	November, 2002	LBG, NACD 2003-1 Beta
9000-6407-27	April, 2003	SRR, NACD 2003-1
9000-6407-28	December, 2003	LBG, NACD 2004-1 Beta
9000-6407-29	April, 2004	SRR, NACD 2004-1
9000-6407-30	March, 2005	LBG, Natural Access 2005-1
9000-6407-31	February, 2009	DEH, Natural Access R8.1
64-0508-01	October, 2009	LBG, NaturalAccess R9.0
64-0508-02	December 2009	LBG, NaturalAccess R9.0.1
64-0508-03 Rev A	October 2010	LBG, NaturalAccess R9.0.4
Last modified: 2010-10-12		

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

1. Introduction

The *Dialogic® NaturalAccess™ ISDN Software Developer's Manual* explains how to develop an ISDN-based telephony application using NaturalAccess™ ISDN for NCC API software. It concentrates on building ISDN for NCC API applications that perform call control using the NaturalAccess NaturalCallControl API (NCC API). For information on building applications that interface with the ISDN Software protocol stack at lower layers, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

This document targets developers of telephony and voice applications who are using NaturalAccess. It defines telephony terms where applicable, but assumes that the reader is familiar with basic telephony concepts. It also assumes that the user is 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

Former terminology	Dialogic terminology
NMS OAM System	Dialogic® NaturalAccess™ OAM System
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

Former terminology	Dialogic terminology
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
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. ISDN overview

Integrated Services Digital Network (ISDN)

Integrated Services Digital Network (ISDN) is an international standard for networking a wide range of services, including voice and non-voice services. The network is completely digital, from one end to the other: voice information is digitized and sent in digital form. Signaling information is sent separately from voice information, using a method called common channel signaling (CCS).

ISDN protocols and protocol layering

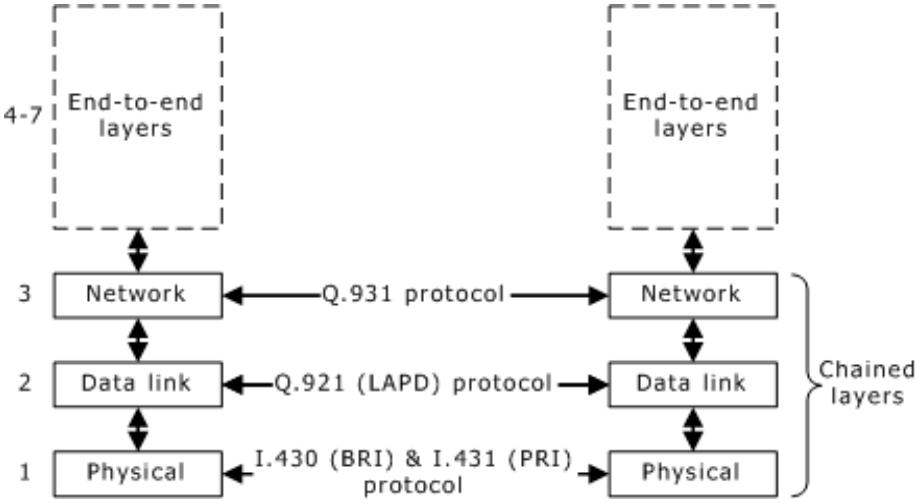
ISDN communications can be described at many levels, from the way bits are transferred from machine to machine to the sets of messages computers pass to one another. A scheme for communication at a certain level is called a protocol.

In the late 1970's, the International Standards Organization (ISO) established the Open Systems Interconnect (OSI) model for communication. ISDN is based on this model. In OSI, seven separate levels, or layers, of communication are defined. The first three layers, called the chained layers, are the lowest levels. The following table describes the chained layers:

Layer	Description
Physical layer (layer 1)	The electrical and mechanical layer. Protocols for this layer describe, from an electrical and mechanical perspective, the methods used to transfer bits from one device to another. A protocol used at this layer is CCITT recommendation I.430/I.431.
Data link layer (layer 2)	The layer above the physical layer. Protocols for this layer describe methods for error-free communication between devices across the physical link. A protocol used at this layer is CCITT recommendation Q.921, also known as Link Access Procedures on the D channel (LAPD).
Network layer (layer 3)	The layer above the data link layer. Protocols for this layer describe methods for transferring information between computers. They also describe how data is routed within and between networks. A protocol used at this layer is CCITT recommendation Q.931.

Layers higher than these are end-to-end layers. They describe how information is exchanged and delivered end-to-end. They also define process-to-process communication, and describe application-independent user services, as well as user interfaces and applications.

The following illustration shows the OSI protocol layering model:



The functionality provided by a layer includes the services and functions of all of the layers below it. A service access point (SAP) is the point at which a layer provides services to the layer directly above it. Each SAP is associated with a unique service access point identifier (SAPI).

ISDN carriers

ISDN is transmitted over standard T1 and E1 carriers. These are typically four-wire digital transmission links. T1 is used mainly in the United States, Canada, Hong Kong, Taiwan, and Japan. E1 is used throughout most of the rest of the world.

Data on a trunk is transmitted in channels. Each channel carries information digitized at 64,000 bits per second (b/s). This topic describes the:

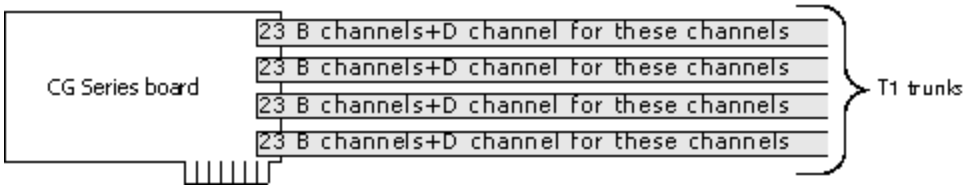
- Primary rate interface (PRI)
- Basic rate interface (BRI)

Primary rate interface (PRI)

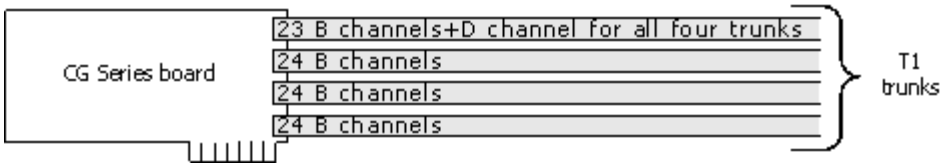
For primary rate ISDN, T1 carries 24 channels. E1 carries 32 channels. The channels are usually used as follows:

- On a T1 trunk, 23 of the 24 channels carry data: voice, audio, data and video signals. These channels are called bearer channels (B channels). On an E1 trunk, 30 of the 32 channels are bearer channels.
- On a T1 or E1 trunk, one channel carries signaling information for all B channels. This is called the D channel.

The following illustration shows a T1 trunk standard configuration:



In setups with multiple T1 ISDN trunks, a non-facility associated signaling (NFAS) configuration is often used. In this configuration, the D channel on one of the ISDN trunks carries signaling for all channels on several other trunks. This leaves channel 24 free on each of the other trunks to be used as another B channel as shown in the following illustration:



Note: NFAS configurations are supported only on T1 trunks. For more information about NFAS, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual.*

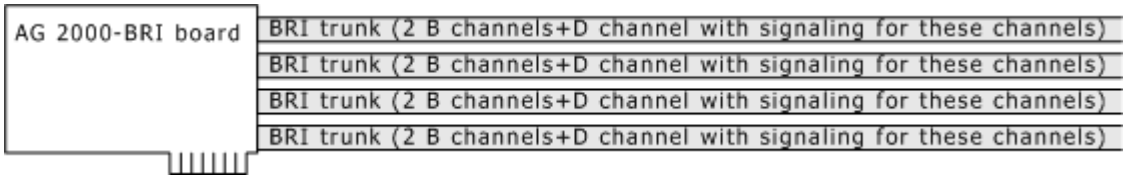
Basic rate interface (BRI)

ISDN is also transmitted over BRI trunks with four-wire digital transmission links. BRI trunks are used mainly in Europe and Asia and transmit data in three channels.

The three channels are usually used as follows:

- Two of the channels are B channels, carrying data: voice, audio, data and video signals at 64000 b/s.
- One of the channels is a D channel, carrying signaling information for the B channels at 16000 b/s.

The following illustration shows a BRI trunk standard configuration:



ISDN Software configurations

ISDN Software allows you to write NaturalAccess applications that communicate with BRI, T1, or E1 trunks to perform voice processing functions and call control using ISDN common channel signaling protocols.

ISDN Software uses one or more digital trunk interface boards as the physical interface to trunk lines. In addition to line interfaces, these boards also feature on-board digital signal processing (DSP) resources that can handle most of the call control and voice processing tasks.

ISDN Software product configurations

Using ISDN Software, you can access ISDN services in three ways:

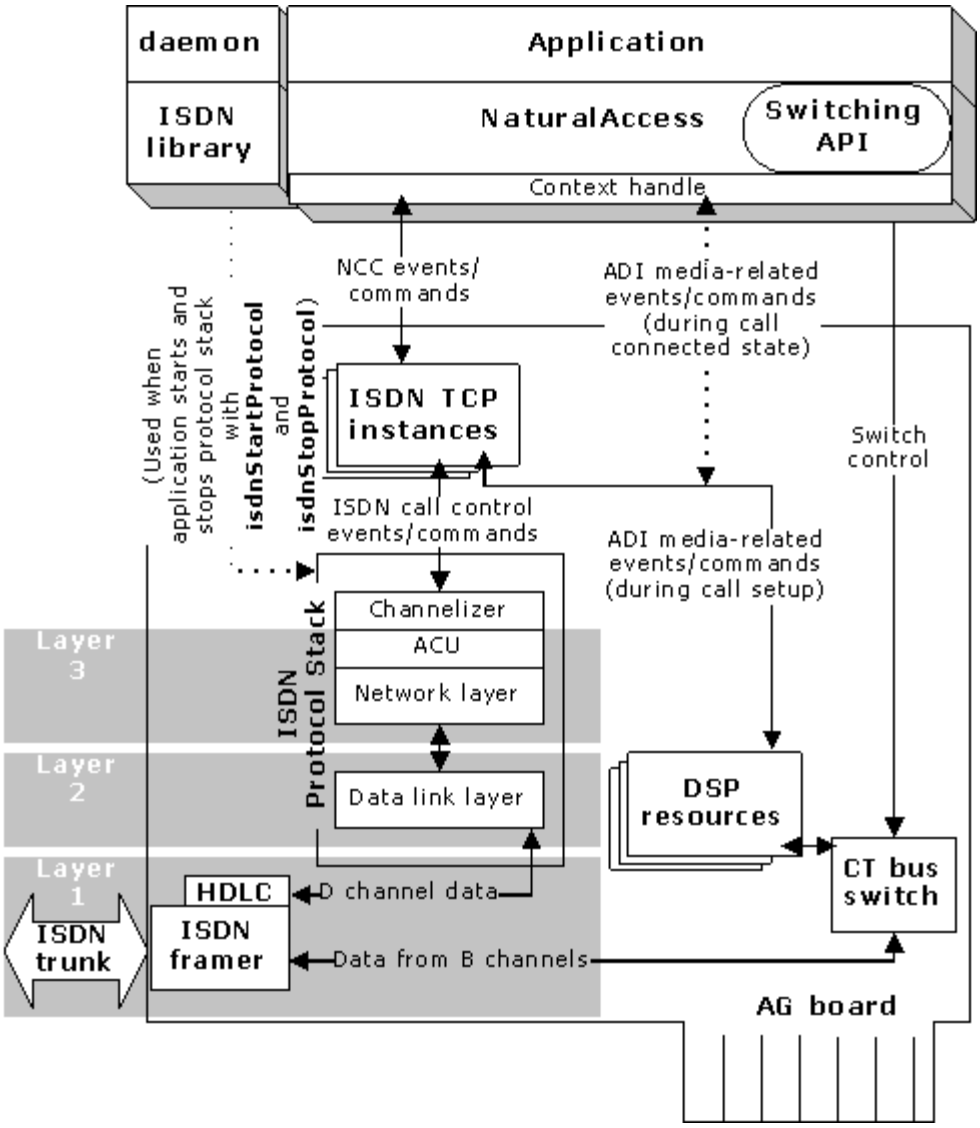
Configuration type	Description
Channelized	Configure ISDN Software so an application can perform call control and other operations using the NCC API. This is the interface described in this manual.
ACU	Access ISDN APIs at the ACU SAPI. Using the ISDN Messaging API, an application can perform a wide range of Q.931 ISDN D channel functions. For more information about this configuration, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.
LAPD	Access ISDN APIs at the data link layer (Layer 2). Using the ISDN Messaging API, an application can send and receive I-frame data in LAPD messages. This data typically consists of Q.931 messages. For more information about this configuration, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

Specify the configuration to use when initializing the ISDN protocol stack, as described in Initializing the application.

ISDN Software channelized configuration

ISDN Software allows you to access ISDN call control services using the NaturalCallControl API (NCC API). ISDN Software provides access through standard function calls. The same function calls are used with any other trunk protocols (such as CAS or DPNSS). The ISDN Software NaturalCallControl interface is described in this manual.

The following illustration shows the ISDN Software application architecture in a channelized configuration:



In a channelized configuration, the ISDN protocol stack and a trunk control program (TCP) run on the board.

Component	Description
ISDN protocol stack	One or more instances of this stack run on the board, one for each D channel. In the ISDN Software channelized configuration, the protocol stack runs in channelized stack mode. In this mode, the stack implements all ISDN layer 2 and layer 3 functionality. When signaling messages for a particular B channel arrive on the D channel, the channelizer within the stack routes them to the TCP associated with that channel. This description also applies to DPNSS.
Trunk control program (TCP)	A TCP serves as the call control interface between NaturalAccess and a B channel on a trunk. At initialization, the application creates a context/TCP instance pair for each B channel (voice channel). The application communicates with the TCP through the context. The TCP for a B channel serves as the interface between the application and the ISDN protocol stack for all signaling for that B channel. It translates NaturalAccess commands involving signaling for the B channel into ISDN messages, and sends them to the ISDN protocol stack, which controls the D channel. It interprets signaling messages coming from the ISDN protocol stack, translates them into appropriate events, and sends them to the application through the context. During call setup, the TCP has exclusive use of the DSP resources allocated to the B channel. Once the call is in the connected state, the application also has direct control of the DSP resources for the channel. For DPNSS, the mechanism described for ISDN also applies.

Note: DSP resources are temporarily available to the application during call setup if the application elects to play user audio while accepting or rejecting a call.

In channelized configuration, ISDN protocol stack instances are started and stopped using two functions from the ISDN Software library. No other functions from this library are used in this configuration. A daemon program included with the ISDN Software can start the stack before ISDN applications are launched, and stop the stack after they are shut down.

B channel information is routed to the DSP resources through the board's CT bus switch. The switch has default behavior, described in Making switch connections for ISDN Software. Alternatively, the switch can be controlled using the NaturalAccess Switching API.

ISDN Software components

ISDN is implemented differently around the world. For this reason, Dialogic provides several variants of its ISDN Software for different countries or regions. The package for a variant contains the software modules you need to allow a board to communicate on a T1, E1, or BRI trunk in the countries that use that variant.

The ISDN Software package for a given variant contains:

- A *readme* file.
- ISDN Software function libraries for NaturalAccess.
- Header files.
- Downloadable object modules containing the ISDN protocol stack software.
- Board keyword files.
- Demonstration programs, with their source code files and makefiles.
- A trunk control program (TCP).
- Several parameter files, in binary format (*.pf* files) and in ASCII format (*.par* files). These files contain ISDN Software parameters. Certain parameters in these files are set to values specific to a certain country or variant.

readme file

The readme file is an ASCII text file that contains release information that does not appear in other documentation. The file is named *readme_isdn.txt*. Refer to this file to learn where the ISDN Software components are located after installation.

ISDN Software function libraries

The ISDN Software function libraries run on the host computer. The application program uses them to interact with the ISDN protocol stacks running on the board and to communicate with the NCC API. The following table lists the NaturalAccess library names for Windows and UNIX:

Operating system	Libraries
Windows	*isdnapi.lib, isdnapi.dll, nccisdn.lib, nccisdn.dll*
UNIX	*libnccisdn.so, libisdnapi.so*

Header files

The following header files are supplied with ISDN Software and are used in a channelized configuration:

File name	Description
isdnval.h	Defines for Q.931 messages created by the stack.
isdndef.h	Event code definitions and ISDN API function prototypes.
isdnparm.h	Parameter structure definitions and manifest constants for parameter structure fields.
isdntype.h	Type definitions, basic and derived types, and entity identifiers.
nccxadi.h	NCC parameter structures.

File name	Description
nccadi.h	NCC values for mediamask, connectmask, and disconnectmask.
nccxisdn.h	ISDN parameter structures and values for the NCC API.

ISDN protocol stack downloadable object modules

The downloadable object module files contain the basic low-level software that a board requires to support ISDN. The modules are transferred from the host into on-board memory when the board boots.

Different module files are supplied for different configurations. The file you use depends upon the board type you are using. For more information about downloadable object modules, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual*.

Board keyword files

Use a board keyword file to specify configuration information for all boards in a system. This information includes whether a board performs switching, which board is the clock master, and which software modules to transfer to the board's memory on startup including which TCPs to load. A board keyword file also contains country-specific information and defines what trunks are assigned to which D channels.

Several sample board keyword files are provided that describe ISDN Software configurations for different boards. Use these files to create a file that describes the system hardware and software setup. For more information, refer to the *Dialogic® NaturalAccess™ ISDN Software Installation Manual.*

Use the NaturalAccess OAM API utilities to assign parameters to a board. The OAM API is installed with NaturalAccess. For more information about OAM, see the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.

Demonstration programs

Several demonstration programs are included with their source code files and makefiles. For more information, see Demonstration programs overview.

Trunk control program (TCP)

isd0.tcp is used with the ISDN Software in channelized configuration. This TCP is also used with the DPNSS protocol stack.

The TCP is transferred to on-board memory when the board is started. An instance of the TCP is associated with each context. A TCP mediates transactions between NaturalAccess, DSP resources, and the protocol stack.

Parameter files

Parameter files contain parameters and values that configure the ISDN Software TCP. Some of these parameters are country-specific. Different values are supplied for them based upon the target country.

Note: Parameter files are useful only if you are configuring the ISDN Software for channelized configuration.

Warning: 	Changing the values of country-specific parameters can affect the regulatory approvals in the target country.

For more information about parameters and how to load and change them, refer to the ISDN TCP parameters overview.

The following table lists the parameter files relevant to your application if you are using the NCC API for call control:

File type / name	Description
*nccxadi**cty**.pf* *nccstart**cty**.pf* ***cty*** is the three character code of the target country. For example, the code for Australia is aus. The versions of these files for Australia are *nccxadiaus.pf* and *nccstartaus.pf*.	Binary parameter files containing a set of country-specific values for NCC API parameters. Most of the values in these files should not be changed. Changing certain values may affect the regulatory approvals in the target country.
nccxisdn.pf	A binary parameter file containing a set of ISDN Software parameters and default values. Changing these parameters directly affects all control messages (for example, messages associated with NCC functions). To change them correctly, you must have knowledge of the ISDN specifications for the target country.
*nccxadi**cty**.par* *nccstart**cty**.par* *nccxisdn.par*	ASCII versions of *nccxadi**cty**.pf, nccstart**cty**.pf*, and *nccxisdn.pf.*

Other components

In addition to the ISDN Software, you need the following components to build an ISDN protocol application:

- One or more T1, E1, or BRI trunk interface boards.
- NaturalAccess.

Warning: 	Dialogic obtains board-level approvals certificates for supported countries. Some countries require that you obtain system-level approvals before connecting a system to the public network. To learn what approvals you require, contact the appropriate regulatory authority in the target country.

NaturalAccess

NaturalAccess is a complete development environment for telephony applications that must be installed to build an ISDN Software application.

For more information, refer to NaturalAccess environment.

NaturalCallControl under NaturalAccess

The NaturalCallControl API (NCC API) features a protocol-independent API, separate call and line state machines, call hold/retrieve capabilities, and the ability to query a protocol's capabilities. Use this service to perform call control operations.

For more information, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

Developing an ISDN Software application

Complete the following steps to create an ISDN Software application:

Step	Action	Where step is documented
1	Install digital trunk interface boards in a system and any other boards you will need for the application.	The installation manuals for the boards.
2	Install NaturalAccess.	The NaturalAccess installation booklet.
3	Install the ISDN Software for each target country where the application will be used.	The *Dialogic® NaturalAccess™ ISDN Software Installation Manual.*
4	Edit the board keyword file so it describes all boards in the system.	The *Dialogic® NaturalAccess™ ISDN Software Installation Manual,* the installation manuals for the boards, and the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.
5	Test your hardware installation.	The installation manuals for the boards.

Step	Action	Where step is documented
6	Write the application.	This manual, the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*, and the NaturalAccess documentation set.

3. ISDN Software programming model

NaturalAccess environment

To build applications using ISDN Software, NaturalAccess must be installed on the system.

NaturalAccess telephony functions are subdivided into groups of logically related functions, called services. Each service has a standard API.

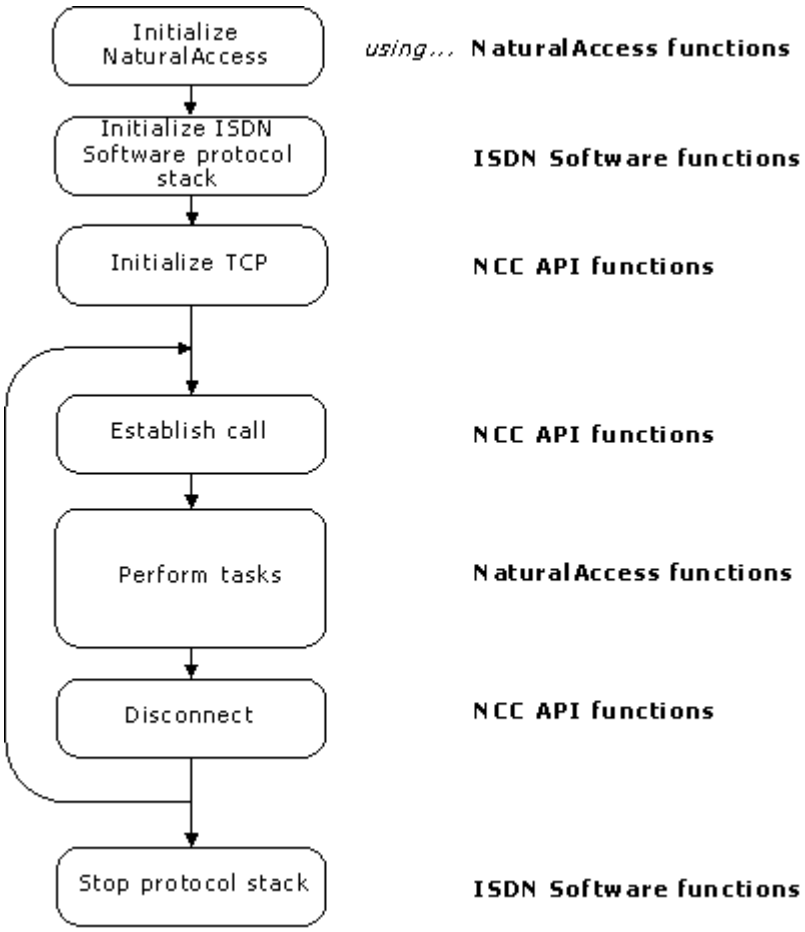
A context organizes services and accompanying resources around a single processing context. A context usually represents an application instance controlling a single telephone call. A service can only be opened once on a context.

An event queue is the communication path from a service to an application. A service generates events indicating certain conditions or state changes. An application retrieves the events from the event queue.

For more information about NaturalAccess, see the *Dialogic® NaturalAccess™ Software Developer's Manual*.

ISDN Software application overview

The following flowchart shows the tasks that are typically performed by an ISDN Software application:



The following table describes each of these tasks:

In this phase...	The application...
Initialize NaturalAccess	Makes switch connections to route D channel data to the HDLC controller and to route B channel information to DSP resources (if necessary). Initializes NaturalAccess services and creates one context for each B channel. The application also creates a dummy context for each D channel.
Initialize ISDN Software protocol stack	Invokes **isdnStartProtocol** to start an ISDN or DPNSS protocol stack instance on each dummy context.
Initialize TCP	Uses **nccStartProtocol** to start a trunk control program (TCP) on each B channel context, loading country specific parameters if necessary.
Establish call	Uses NCC service call control functions to place outgoing calls, receive incoming calls, or both.
Perform tasks	Uses functions from NaturalAccess or other APIs to play or record voice, generate or detect DTMF tones, send and receive faxes, and other tasks.
Disconnect call	Uses NCC service call control functions to disconnect and release the call. The application then establishes another call, or stops the protocol stack.
Stop protocol stack	Invokes **isdnStopProtocol** to stop the ISDN or DPNSS protocol stack.

Initializing the boards

Before you can run an ISDN Software application, you must initialize and load DSP files, TCPs, and protocol stack runfiles to the boards. The items to load to the boards are specified in a board keyword file.

To load the components to your boards, run *oamsys*, the board initialization and monitoring utility. This utility reads the board keyword file and sets up your boards as described in the file.

For more information, refer to the *Dialogic® NaturalAccess™ ISDN Software Installation Manual* and the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.

Receiving ISDN events

All messages and events from the ISDN Software protocol stack are returned to the application through the standard NaturalAccess event handling mechanism, along with standard NaturalAccess events and any events specific to NaturalAccess extensions. They arrive in the form of the standard CTA_EVENT data structure. For more information about NaturalAccess events, refer to the *NaturalAccess Developer's Reference Manual*.

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

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

Some call control-related NaturalAccess events are unsolicited. They can occur at any time, regardless of the application's current activities. Usually, unsolicited events indicate that something has happened on the trunk. For a summary of these events, refer to Unsolicited events.

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

4. Initializing an ISDN application

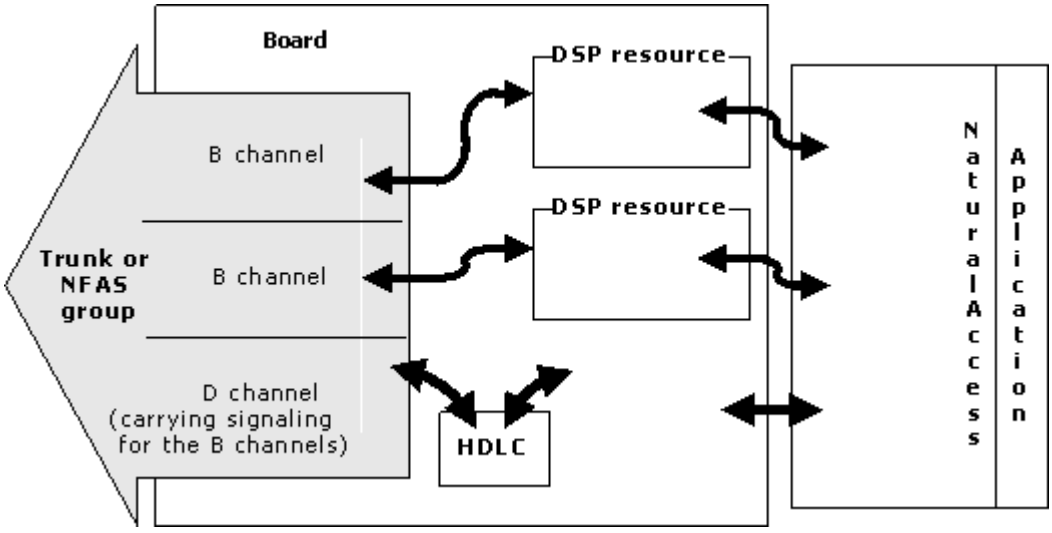
Initializing the application

An ISDN Software application performs the following initialization tasks for each trunk or NFAS group:

1. Make switch connections
2. Initialize NaturalAccess and create contexts
3. Initialize an ISDN or DPNSS protocol stack instance
4. Start a TCP on each B channel context

Make switch connections

If necessary, the application makes switch connections to route D channel data to the HDLC controller, and to route B channel information to DSP resources. Certain default connections are automatically made if switching is not enabled. The following illustration shows routing channel data to on-board resources:



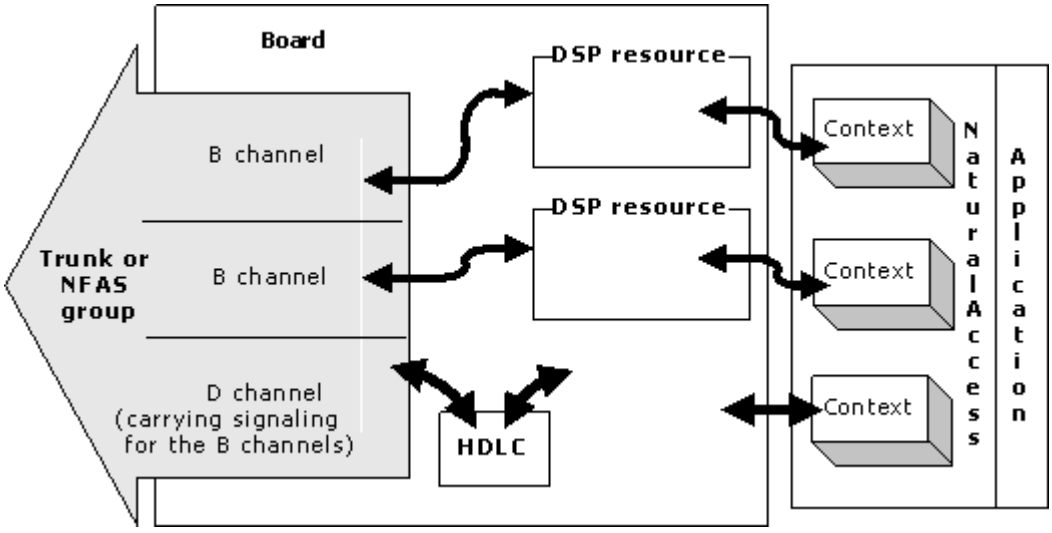
For more information, see Making switch connections for ISDN Software.

Initialize NaturalAccess and create contexts

The application initializes NaturalAccess, and creates a separate context for each B channel on the trunk or NFAS group that it will interact with. For more information, see Initializing NaturalAccess.

The application must also create a separate dummy context for the D channel. This context is used only to refer to the stack instance on the channel with other ISDN Software library functions (such as **isdnStopProtocol**). Signaling from a D channel is not accessed through this context; instead, the channelizer in the protocol stack routes the signaling for each B channel to the context for that B channel.

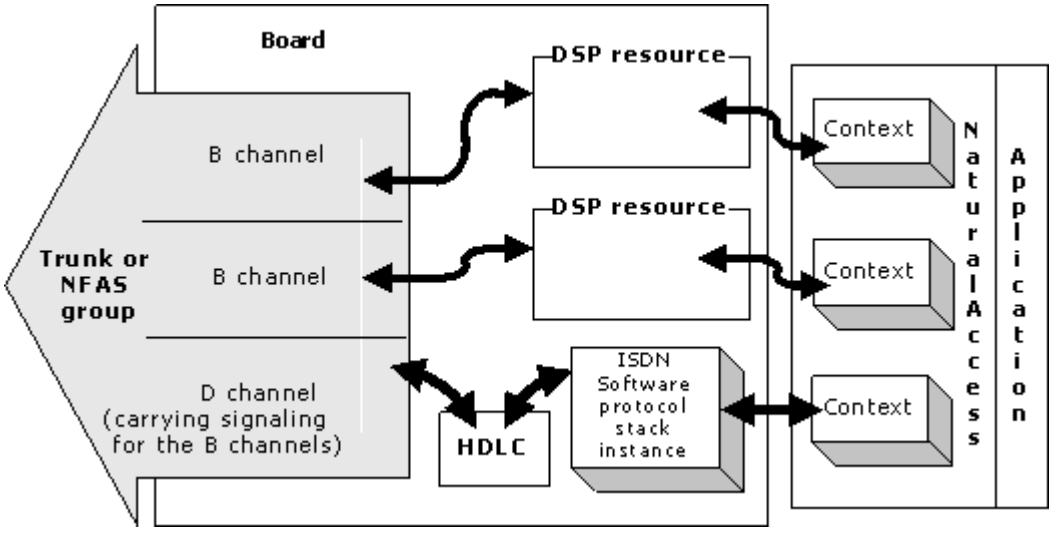
The following illustration shows creating contexts for channels:



Initialize an ISDN or DPNSS protocol stack instance

The application initializes an ISDN or DPNSS protocol stack instance on the D channel context using **isdnStartProtocol**. This function starts up an ISDN protocol stack instance on the dummy context in channelized stack mode. In the function invocation, the trunk is specified using its network access identifier (NAI) and NFAS group number (for duplicate NAI values).

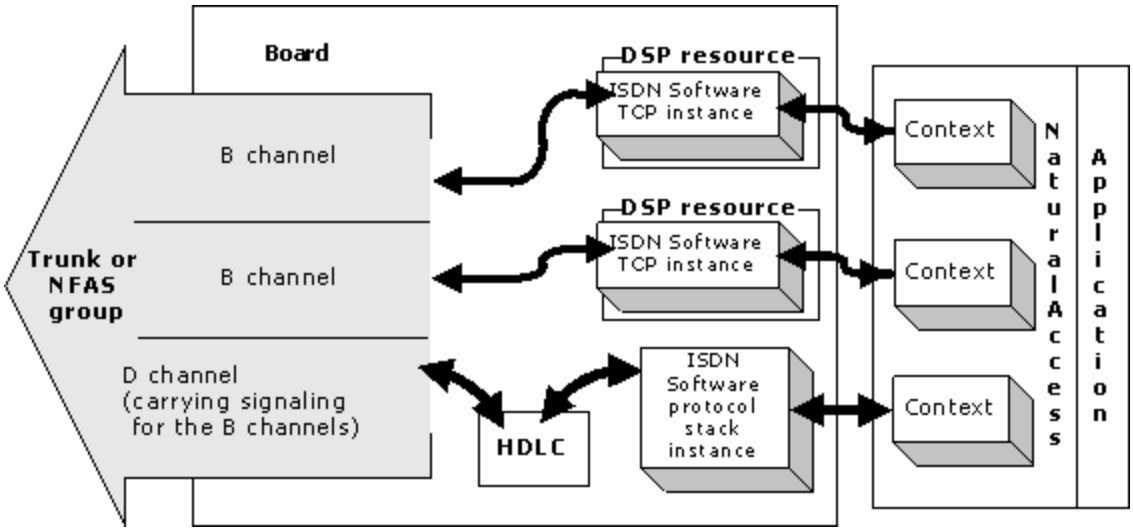
The following illustration shows calling isdnStartProtocol:



For more information, see Setting up D channels and ISDN TCP parameters overview.

Start a TCP on each B channel context

The application starts a TCP on each B channel context, loading country-specific parameters for each context (if necessary). The following illustration shows starting TCPs on B channel contexts:



For more information, see Starting ISDN TCP instances.

Making switch connections for ISDN Software

To allow an application access to ISDN channels, several switch connections must be made. In the board keyword file, if Clocking.HBus.ClockMode=StandAlone, these settings are automatically made when the board boots.

If Clocking.HBus.ClockMode is set to any other value, the application must set these values using the NaturalAccess Switching API or *swish* utility.

The application must make the following connections:

- On each trunk, streams and timeslots carrying D channel information to and from the HDLC controller (if any) must be connected to streams and timeslots accessible by the ISDN protocol stack. These connections must be full duplex. They must be made before the ISDN protocol stack is initialized. Initialization is described in Initializing an ISDN Software protocol stack instance.
- The streams and timeslots carrying voice information to and from the trunk must be connected to the streams and timeslots carrying voice information to and from the DSP resources. The connections must be full duplex.

The connections differ depending upon whether the NetworkInterface.T1E1.SignalingType keyword is set to PRI or RAW. Specifically, if NetworkInterface.T1E1.SignalingType=RAW, no connections are made between the HDLC controller and signaling streams. This setting is for trunks that are included in NFAS groups and do not have a D channel in operation. For more information about these keywords and about NFAS, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual.*

Connections are listed in MVIP-95 nomenclature unless otherwise specified.

Note: Because on CG boards framer signaling is hard wired to the HDLCs, you cannot switch framer signaling to the HDLCs.

The following table lists the default connections:

Board	Default connections
Four-trunk T1	**If NetworkInterface.T1E1[x]. SignalingType=PRI** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..22 => 17:0..22, 16:0..22 => 1:0..22 Trunk 2: 4:0..22 => 17:24..46, 16:24..46 => 5:0..22 Trunk 3: 8:0..22 => 17:48..70, 16:48..70 => 9:0..22 Trunk 4: 12:0..22 => 17:72..94, 16:72..94 => 13:0..22 Full duplex, between HDLC controller and signaling streams (AG boards only): Trunk 1: 2:0 => 21:0, 20:0 => 3:0 Trunk 2: 6:0 => 23:0, 22:0 => 7:0 Trunk 3: 10:0 => 25:0, 24:0 => 11:0 Trunk 4: 14:0 => 27:0, 26:0 => 15:0 **If NetworkInterface.T1E1[x]. SignalingType=RAW** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..23 => 17:0..23, 16:0..23 => 1:0..23 Trunk 2: 4:0..23 => 17:24..47, 16:24..47 => 5:0..23 Trunk 3: 8:0..23 => 17:48..71, 16:48..71 => 9:0..23 Trunk 4: 12:0..23 => 17:72..95, 16:72..95 => 13:0..23

Board	Default connections
Four-trunk E1	**If NetworkInterface.T1E1[x]. SignalingType=PRI** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..29 => 17:0..29, 16:0..29 => 1:0..29 Trunk 2: 4:0..29 => 17:30..59, 16:30..59 => 5:0..29 Trunk 3: 8:0..29 => 17:60..89, 16:60..89 => 9:0..29 Trunk 4: 12:0..29 => 17:90..119, 16:90..119 => 13:0..29 Full duplex, between HDLC controller and signaling streams (AG boards only): Trunk 1: 2:0 => 21:0, 20:0 => 3:0 Trunk 2: 6:0 => 23:0, 22:0 => 7:0 Trunk 3: 10:0 => 25:0, 24:0 => 11:0 Trunk 4: 14:0 => 27:0, 26:0 => 15:0 **If NetworkInterface.T1E1[x]. SignalingType=RAW** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..30 => 17:0..30, 16:0..30 => 1:0..30 Trunk 2: 4:0..30 => 17:31..61, 16:30..61 => 5:0..30 Trunk 3: 8:0..30 => 17:62..92, 16:60..92 => 9:0..30 Trunk 4: 12:0..30 => 17:93..123, 16:90..123 => 13:0..30 NetworkInterface.T1E1[x]. SignalingType is usually set to RAW only for trunks that are included in NFAS groups and do not have a D channel in operation. NFAS is not supported on E1 trunks, so you would probably not use this configuration.

Board	Default connections
Two-trunk T1	**If NetworkInterface.T1E1[x]. SignalingType=PRI** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..22 => 17:0..22, 16:0..22 => 1:0..22 Trunk 2: 4:0..22 => 17:24..46, 16:24..46 => 5:0..22 Full duplex, between HDLC controller and signaling streams (AG boards only): Trunk 1: 2:0 => 21:0, 20:0 => 3:0 Trunk 2: 6:0 => 23:0, 22:0 => 7:0 **If NetworkInterface.T1E1[x]. SignalingType=RAW** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..23 => 17:0..23, 16:0..23 => 1:0..23 Trunk 2: 4:0..23 => 17:24..47, 16:24..47 => 5:0..23
Two-trunk E1	**If NetworkInterface.T1E1[x]. SignalingType=PRI** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..29 => 17:0..29, 16:0..29 => 1:0..29 Trunk 2: 4:0..29 => 17:30..59, 16:30..59 => 5:0..29 Full duplex, between HDLC controller and signaling streams (AG boards only): Trunk 1: 2:0 => 21:0, 20:0 => 3:0 Trunk 2: 6:0 => 23:0, 22:0 => 7:0 **If NetworkInterface.T1E1[x]. SignalingType=RAW** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..30 => 17:0..30, 16:0..30 => 1:0..30 Trunk 2: 4:0..30 => 17:30..61, 16:30..61 => 5:0..30 NetworkInterface.T1E1[x]. SignalingType is usually set to RAW only for trunks that are included in NFAS groups and do not have a D channel in operation. NFAS is not supported on E1 trunks, so you would probably not use this configuration.

Board	Default connections
Four trunk BRI	**If NetworkInterface.T1E1[x]. SignalingType=PRI** Full duplex, between trunk voice information and DSP resources: Trunk 1: 0:0..1 => 5:0..1, 4:0..1 => 1:0..1 Trunk 2: 0:2... 3 => 5:2..3, 4:2..3 => 1:2..3 Trunk 3: 0:4..5 => 5:4..5, 4:4..5 => 1:4..5 Trunk 4: 0:6..7 => 5:6..7, 4:6..7 => 1:6..7 **If NetworkInterface.T1E1[x]. SignalingType=RAW** None.

The *isdncta* demonstration program shows how an application makes these connections.

Initializing NaturalAccess

To begin call control operations, the application performs the following tasks:

Task	Description
1	Initializes NaturalAccess services (including the ISDN Software) using **ctaInitialize**.
2	Creates one or more event queues, using **ctaCreateQueue**. Each function call creates a queue and returns a handle. Make sure at least one queue is attached to the ADI API manager.
3	Creates one or more contexts using **ctaCreateContext**. Each call creates a context and returns a context handle.
4	Opens services on the contexts using **ctaOpenServices**.

Note: If D channel backup is defined in your configuration, do not initialize NaturalAccess on the context bearing the backup D channel. See the *Dialogic® NaturalAccess™ ISDN Software Installation Manual* for more information on D channel backup.

For more information about initializing NaturalAccess, see the *Dialogic® NaturalAccess™ Software Developer's Manual.*

Specifying B channel contexts

Create a separate context for each B channel on each trunk or NFAS group your application will interact with. To open contexts under NaturalAccess, use **ctaCreateContext** followed by **ctaOpenServices**. In each call to **ctaOpenServices**, specify ***stream***, ***timeslot***, and ***mode.***

stream

For B channel contexts, set ***stream*** to the voice streams for the on-board DSPs. These streams are:

Board	DSP resource stream
AG 2000-BRI	MVIP 95: Local stream 0
All other digital trunk interface boards	MVIP 95: Local stream 16

timeslot

For B channel contexts, set to a base timeslot in ***stream***.

mode

For B channel contexts, set ***mode*** to ADI_VOICE_DUPLEX. This mode allows voice (inband) transmission and reception. Because ISDN signaling is not carried in the B channels, do not use the modes involving signaling defined in the ADI documentation.

Set these values in the CTA_MVIP_ADDR structure passed to **ctaOpenServices**.

Specifying dummy D channel contexts

Create a separate dummy context for each D channel on each trunk or NFAS group the application will interact with. The context for a D channel is used only to refer to the stack instance on the channel in other ISDN Software library function calls (such as **isdnStopProtocol**). Signaling from a D channel is not accessed through the dummy context. The channelizer in the protocol stack routes the signaling for each B channel to the context for that B channel.

The following table shows how the number of trunks on a board is related to the number of D channels available.

Boards with...	Can support up to...
Four trunks	Four separate D channels.
Two trunks	Two separate D channels.

To open dummy contexts under NaturalAccess, use **ctaCreateContext** followed by **ctaOpenServices**. In each call to **ctaOpenServices**, ***stream***, ***timeslot*** and ***mode*** should be set to 0, since no DSP processing resources are needed to control the D channel data stream.

Set these values in the CTA_MVIP_ADDR structure passed to **ctaOpenServices**.

Setting up D channels

Initialize an ISDN protocol stack instance on each D channel context, using **isdnStartProtocol**. This function starts up an ISDN protocol stack instance on the dummy context in channelized stack mode.

Note: If D channel backup is defined in the configuration, do not initialize NaturalAccess on the context bearing the backup D channel. See the *Dialogic® NaturalAccess™ ISDN Software Installation Manual for* more information on D channel backup.

Network access identifiers (NAIs)

A trunk is referenced by a numeric value, called a network access identifier (NAI). All NAIs for a given board are defined in the board keyword file. NAI values are used as interface identifier values inside ISDN Q.931 messages sent across the network. When you initialize an ISDN protocol stack instance for a context (using **isdnStartProtocol**), you specify the NAI of the trunk to associate with the context. From then on, the application communicates with the D channel on that trunk through the context handle. For example, when an event is received, the context handle indicates the trunk on which the event occurred.

In most configurations, NAIs are unique for each trunk, so the NAI value can be used to refer to a specific trunk on a particular board.

Some variants support duplicate NAI values, where NAIs are not unique across NFAS groups or the board. For these configurations, the application must provide both the NAI and NFAS group number to refer to a trunk when calling **isdnStartProtocol**.

Different board types support different numbers of D channels, with different NAIs.

The following table shows what each board type supports:

Digital trunk interface board type	Number of D channels	NAIs	Number of NFAS groups
Four trunk boards	Up to 4	0 through 3	Up to 4
Two trunk boards	Up to 2	0 through 1	Up to 2

The NAI and NFAS group number of a trunk are specified in the board keyword file. For more information, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual*.

Initializing an ISDN Software protocol stack instance

To initialize an ISDN Software or DPNSS protocol stack instance on a context, use one of the following methods:

- Run the ISDN daemon (*isdncta*).
- Invoke **isdnStartProtocol**.

Initializing a stack instance using the ISDN Software daemon

To initialize an ISDN protocol stack instance, run the *isdncta* daemon supplied with ISDN Software. This daemon starts the ISDN protocol stack, and also makes the switch connections needed to support ISDN Software. When an ISDN protocol stack instance is initialized with the daemon, applications that use the NCC API will operate with the ISDN TCP in the same way they would operate with any other TCP.

The daemon initializes an ISDN protocol stack instance by calling **isdnStartProtocol**. On the command line, indicate the NAI to use and, if multiple CCID is configured, the NFAS group number for this NAI. To learn how to launch *isdncta*, see isdncta (ISDN daemon).

Initializing a stack instance using isdnStartProtocol

To initialize an ISDN protocol stack instance from within your application, call **isdnStartProtocol**.

Note: Once you reference the context in **isdnStartProtocol**, do not reference that context in any other function call except **isdnStopProtocol** (to stop the stack).

To direct **isdnStartProtocol** to initialize a stack instance in channelized stack mode, set the ***protocol***, ***partner_equip***, and ***parms*** arguments as described in the following table. For DPNSS, refer to **isdnStartProtocol** in the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

<table>
<tr><th>Argument</th><th>Set to...</th></tr>
<tr><td>protocol</td><td>ISDN_PROTOCOL_CHANNELIZED</td></tr>
<tr><td>partner_equip</td><td>The type of equipment connected to the board.
<table>
<tr><th>Set the value to...</th><th>If the board is...</th></tr>
<tr><td>EQUIPMENT_NT</td><td>Connected to network equipment.</td></tr>
<tr><td>EQUIPMENT_TE</td><td>Acting as network equipment.</td></tr>
</table></td></tr>
<tr><td>parms</td><td>Pointer to a parameter structure to configure the stack. If the application needs to change any of the parameters for Natural Call Control, it should pass the structure name ISDN_PROTOCOL_PARMS_CHANNELIZED in this call. This structure is identical to the ISDN_PROTOCOL_PARMS_ Q931CC structure.

The behavior bits that govern the stack's automatic responses (such as CC_SEND_CALL_PROC_RQ) must not be modified in channelized configuration. The only exceptions to this rule are the behavior bits governing Overlapped sending and receiving.

Refer to the Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual for more information on ISDN Software parameters.

If the application will not change parameters, pass NULL to accept the default settings. The default parameters for the channelized stack mode enable the required service access points (SAPIs).</td></tr>
<tr><td>n</td><td>Where n is an unsigned integer representing the NFAS group number, used for duplicate NAI values only.

Note: Multiple CCID must be enabled by setting enable_Multiple_CCID to 1 in parms.</td></tr>
</table>

Three other structure members in the ISDN_PROTOCOL_PARMS_CHANNELIZED structure are important when calling **isdnStartProtocol** to initialize the ISDN stack:

Structure member	Action
NS_EXPLICIT_INTERFACE_ID	Forces the interface identifier value to be sent in output call control messages (SETUP, PROCEEDING, and others). Applies to the USA variants.
NS_SEND_USER_CONNECT_ACK	Forces a CONNECT_ACK message to be sent by the TE side upon receipt of a CONNECT message. Applies to the ETSI and Europe variants.
NS_PRESERVE_EXT_BIT_IN_CHAN_ID	Applicable when the variant is DMS, USA, or incoming call. If set, the channel ID's octet 3.3's extension bit is set to the value received in a SETUP message inside the PROCEEDING or ALERT messages.

For information about behavior bits and **isdnStartProtocol**, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

The ISDNEVN_START_PROTOCOL event contains the completion status of the start request. If the ISDN protocol stack instance started successfully, the value field in this event contains SUCCESS. Otherwise, another value appears.

Starting ISDN TCP instances

When all ISDN protocol stack instances have been created, the application starts an ISDN TCP instance for each B channel context. Complete the following steps to start a TCP instance:

Step	Action
1	The NaturalAccess Parameter Management service loads the ISDN Software TCP parameters and default values.
2	The application modifies the values, if necessary.
3	The application calls **nccStartProtocol** to start the TCP instance.

Loading parameters

When you install ISDN Software, parameter files are installed. The parameters in these files configure the TCP.

Some parameters determine the amplitude, frequency, length and pattern of the busy tone, ring tone, and reorder tone. Since the tones differ from country to country, these parameters are country-specific, and should not be modified. Other parameters determine the service to request when placing an outbound call, the mode the TCP runs in, the channel direction, and the signal sending mask. These parameters can be modified to suit your application.

For information about loading and changing parameters, see ISDN TCP parameters overview.

Starting a TCP on a context

Once a context is opened and the TCP parameters are loaded, the application starts a TCP on that context using the loaded parameters. Once a TCP has started on a context, the application can use call control functions to place and answer calls on that context.

Note: To start a TCP from within an application, the TCP must have been downloaded to the board at system initialization time. The *oamsys* program downloads all TCPs specified in the *oamsys.cfg* file. For information about the configuration file, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual* and the *Dialogic® NaturalAccess™ OAM System Developer's Manual.*

nccStartProtocol starts a TCP on a context. For ISDN Software call control, ***protname*** passed to **nccStartProtocol** is set to isd0. ***protstartparms*** is NULL, so the default TCP parameters are used, loaded as described in ISDN Software parameter files.

When **nccStartProtocol** is called, NCCEVN_STARTPROTOCOL_DONE is returned. If the TCP is started successfully, the event value field contains CTA_REASON_FINISHED. Otherwise, the value field contains another reason code.

Stopping an ISDN protocol stack instance

To stop an ISDN or DPNSS protocol stack instance, use one of the following methods:

- Stop the ISDN daemon (*isdncta*), if it is running.
- Call **isdnStopProtocol** from within the application that called **isdnStartProtocol**.

Either method shuts down the ISDN protocol stack instance, and releases all on-board resources and buffers formerly used by the stack instance.

The ISDNEVN_STOP_PROTOCOL event contains the completion status of the stop request. If the stack instance stopped successfully, the value field in this event contains SUCCESS. Otherwise, another reason code appears here.

For more information about **isdnStopProtocol**, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

5. ISDN Software call control

ISDN Software and the NCC API

If the NaturalAccess NaturalCallControl API (NCC API) is active, and a protocol was started on a line using **nccStartProtocol**, call control proceeds as described in the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*. This section outlines NCC API call control from an ISDN Software point of view. It contains:

- Descriptions of NCC API functions and associated events, when used with ISDN Software.
- Descriptions of the structures returned by **nccGetCallStatus** and **nccGetExtendedCallStatus**, when ISDN Software is active.
- An explanation of how to send extended call information with call control messages.

Call control operations supported by ISDN Software

ISDN Software supports the following call control operations:

Operation	Supported?
Receiving inbound calls	Yes, for all variants
Placing outbound calls	Yes, for all variants
Releasing calls	Yes, for all variants
Call blocking	Yes, for all variants
Call transfer	Yes, for 4ESS, NI2, DMS, ETSI, Q.SIG, and DPNSS variants only
Call hold/retrieve	Yes, for ETSI, Q.SIG, and DPNSS variants only

NCC API call control model

This topic introduces the NCC API call control model and defines key concepts.

Lines and calls

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 (or 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 (for example, not disconnected or held).

Programmatically, 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.

NCC API events

When performing call control functions, the NCC API processes events: both 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.

Network event type	Description
Transitional	Generated when the call or line state changes. These are generated because the NCC API needs the application to choose an action or acknowledge that an application command is proceeding.
Informational	These 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.

Call control states

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 certain set of occurrences (for example, a specific function call by the application, or actions by the remote party) is defined, which may cause the line or call to change to another state. Whenever a state change occurs, the application is notified by an event.

The NCC API call control model differentiates between line states and call states. A line can be in any of five possible states. A call on the line can be in any of 11 states.

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

Line states

The following table lists the line states, and descriptions for each. For more information, refer to NCC API state machines.

Line state	Description
Uninitialized	Initial state of line. Signifies that a protocol has not been started.
Idle	No active calls currently exist on the line. If there are any calls on the line, they are either held or in disconnected call state. The line is prepared to accept an incoming call or place an outbound call.

Line state	Description
Active	There is at least one active call on this line.
Blocking	All inbound and outbound calls are blocked.
Out of service	The network has placed the line out of service.

If all calls on a line are held or are in a disconnected call state (for example, all calls are inactive), the line state changes to idle. If a call becomes active on the line (for example, a held call is retrieved or a new call comes in or is placed), the line state returns to active.

The following tables list the call states and descriptions for each. For more information, refer to NCC API state machines.

Inbound call states

Call state	Description
Seizure	First indication of a call that is being set up (seized). The event contains a call handle to identify the call for subsequent call-based functions.
Receiving digits (optional)	A set of one or more incoming digits is expected to qualify the incoming call.
Incoming	A call was delivered from the network to the NCC API.
Accepting (optional)	A call was accepted by the application but has not yet been answered or rejected. When a call is in this state, the application can play media functions (such as playing a voice file) before connecting or rejecting the call.
Answering	The NCC API is in the process of answering the call and establishing a connection.
Rejecting	The NCC API is in the process of rejecting the inbound call.

Outbound call states

Call state	Description
Outbound initiated	A call was initiated.
Placing	The line was seized and the network allowed the call to be placed by the NCC API. In other words, glare has been resolved. NCC initiates dialing.
Proceeding	The switch accepted the call setup request and is in the process of attempting to ring the receiving end. Call progress analysis begins.

Connected / disconnected call states

Call state	Description
Connected	A connection exists between the calling parties. The application can use DSP resources if necessary.
Disconnected	A connection no longer exists. A disconnected call is no longer active.

NCC API state machines

The NCC API call control model differentiates between line states and call states. This topic describes NCC API line state and call state machines.

Line states

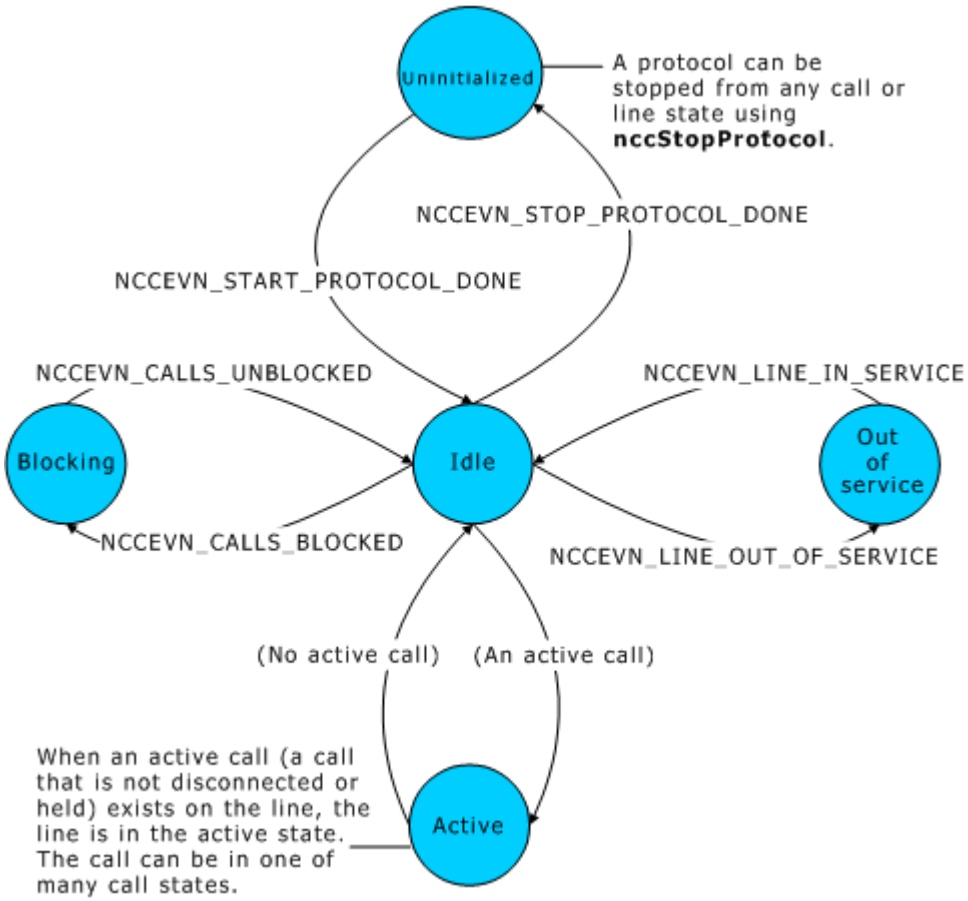
The following table lists the line states. Associated with each state are one or more transitional events, which indicate transition into the state. These are noted in the table.

An application can use **nccGetLineStatus** to determine the state of a line.

Line state	Description
Active	Entered through the NCCEVN_SEIZURE_DETECTED event, when a call is coming in. Also can be entered through the NCCEVN_CALL_RETRIEVED event, which means a call has become active and that the line has reentered the active line state.
Blocking	Entered through the NCCEVN_CALLS_BLOCKED event, solicited by **nccBlockCalls**.
Idle	Entered through the NCCEVN_START_PROTOCOL_DONE event, which is solicited by **nccStartProtocol**. Entered through the NCCEVN_CALLS_UNBLOCKED event, which is solicited by **nccUnblockCalls**. Entered through the NCCEVN_LINE_IN_SERVICE event (an unsolicited event). Entered from the active state, through a NCCEVN_CALL_HELD or NCCEVN_CALL_DISCONNECTED event. These events mean that a call has been placed on hold or is disconnected, and is therefore not active. When no calls are active on a line, the line enters the idle line state.
Out of service	Entered through the NCCEVN_LINE_OUT_OF_SERVICE event, which is an unsolicited event.

Line state	Description
Uninitialized	Initial state of line. When the NCC API is opened on a context, the line handle (signified by the NaturalAccess handle) is created in an uninitialized state. Entered through the NCCEVN_STOP_PROTOCOL_DONE event, which is solicited by **nccStopProtocol**.

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



Call states

The following table lists the call states. Associated with each state are one or more transitional events, which indicate transition into the state. These are noted in the table.

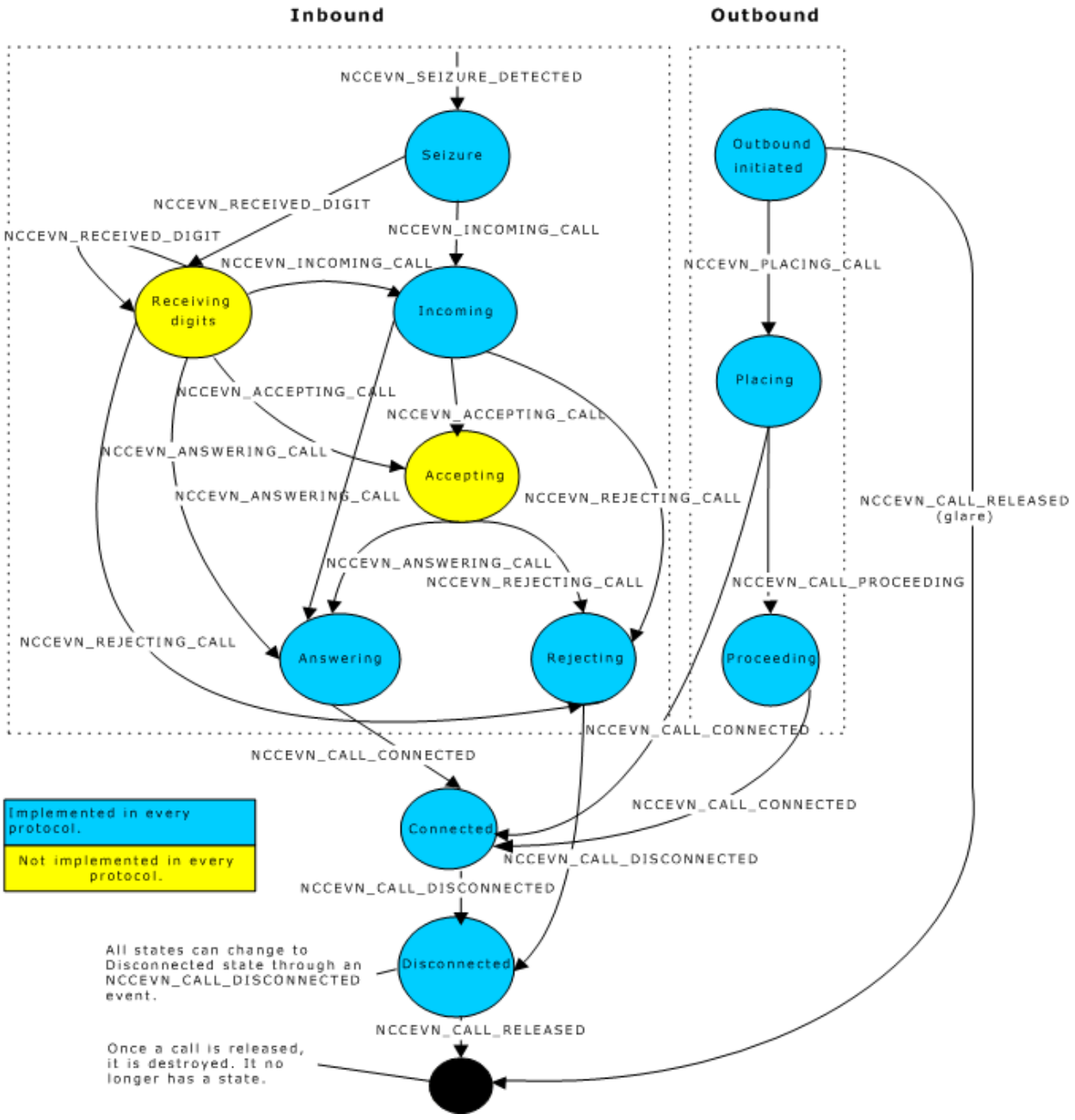
The application can use **nccGetCallStatus** to determine the state of a call.

State	Description
Accepting (optional)	Entered through the 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.

State	Description
Answering	Entered through the NCCEVN_ANSWERING_CALL event in response to invocation of **nccAnswerCall**.
Connected	Entered through an unsolicited NCCEVN_CALL_CONNECTED event after a call was successfully answered by the remote party, or the NCCEVN_CALL_CONNECTED call control connectmask conditions have occurred on an outbound call. For example, connect on proceeding.
Disconnected	Entered from any state through an unsolicited NCCEVN_CALL_DISCONNECTED event. Entered through a 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	Entered through the NCCEVN_INCOMING_CALL event.
Outbound initiated	Entered by attempting to place a call using **nccPlaceCall**. Transition into this state is not indicated by any event.
Placing	Entered through NCCEVN_PLACING_CALL event in response to invocation of **nccPlaceCall**.
Proceeding	An NCCEVN_CALL_PROCEEDING event indicates that the call has entered this state.
Receiving digits (optional)	Entered through an unsolicited NCCEVN_RECEIVED_DIGIT event. This event is generated only if the NCC.START.overlappedreceiving parameter is set. See the *Natural Call Control Service Developer's Reference Manual* for more information. The OVERLAPPED_RECEIVING bit in the capabilitymask returned by **nccQueryCapability** indicates if the protocol supports this state.
Rejecting	Entered through NCCEVN_REJECTING_CALL event in response to invocation of **nccRejectCall**, or as a result of not responding in time to an NCCEVN_INCOMING_CALL.
Seizure	Entered through an unsolicited NCCEVN_SEIZURE_DETECTED event.

Some call states are optional. A call only enters an optional call state, and the event indicating the transition is generated only if the proper parameter enabling or disabling the event is set. For more information about parameters, see the *Natural Call Control Service Developer's Reference Manual*.

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



NCC API functions

ISDN Software invokes NCC API functions to perform call control. For most of these functions, one or more events are returned in response to the invocation.

There are two types of events associated with call control operations:

Event type	Description
Solicited	Signify acknowledgments from the API or the TCP, or occur as a consequence of some function call.
Unsolicited	Can occur at any time, regardless of the application's current activities. Usually, unsolicited events indicate that something has happened on the line.

Certain NCC.START and NCC.X.ADI_ISDN.START parameters dictate whether certain informational call control events are generated. For detailed information about NCC.START parameters, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*. For information about NCC.X.ADI_ISDN.START parameters, refer to ISDN TCP parameters overview.

For information about the NCC API call control-related functions, refer to Call control functions and solicited events.

For detailed documentation of the NCC API functions, parameters, and events, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual.*

Call control functions and solicited events

The NCC API call control-related functions and their associated events are:

- nccAcceptCall
- nccAnswerCall
- nccAutomaticTransfer
- nccBlockCalls
- nccDisconnectCall
- nccGetCallStatus
- nccGetExtendedCallStatus
- nccGetLineStatus
- nccHoldCall
- nccPlaceCall
- nccRejectCall
- nccReleaseCall
- nccRetrieveCall
- nccSendCallMessage
- nccSendDigits
- nccSendLineMessage
- nccSetBilling
- nccStartProtocol
- nccStopProtocol
- nccTransferCall
- nccUnBlockCalls

For detailed documentation of the functions, parameters, and events, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

nccAcceptCall

nccAcceptCall directs the TCP to accept an incoming call without answering it. The application can perform other operations before answering or rejecting the call. The application can accept a call using any of the following methods:

- NCC_ACCEPT_PLAY_RING: Play a ring tone.
- NCC_ACCEPT_QUIET: Accept call, but do not play tone or other audio.
- NCC_ACCEPT_USER_AUDIO: The application plays a recorded message. If remote party disconnects, the TCP interrupts the message.

Associated events	Description
NCCEVN_ACCEPTING_CALL	The TCP is accepting the call, using the specified method. The call state changes to accepting.
NCCEVN_CALL_DISCONNECTED	A DISCONNECT message was received from the network, indicating that the remote party has hung up. The event value field contains the reason. The call state changes to disconnected.
NCCEVN_REJECTING_CALL	The application failed to invoke **nccAnswerCall**, **nccAcceptCall**, or **nccRejectCall** within the period specified in the protocol's NCC.START.waitforpctime parameter. The call is automatically rejected and enters the rejecting call state. The event value field contains NCC_REJECT_HOST_TIMEOUT. A tone is played to the network. The NCC.X.ADI_ISDN.START.flags parameter controls which TCP messages are sent when **nccAcceptCall** is invoked. By default, a PROGRESS message is sent to the network.

nccAnswerCall

Directs the TCP to answer a call after a specified number of rings.

Associated events	Description
NCCEVN_ANSWERING_CALL	The TCP is answering. The call enters the answering call state. An ALERT message is sent to the network. The TCP plays the number of rings specified in the **nccAnswerCall** call. A CONNECT message is sent to the network. By default, **nccAnswerCall** does not send a PROGRESS message.

Associated events	Description
NCCEVN_CALL_CONNECTED	The CONNECT message was acknowledged by the network and the connection is established. The call state changes to connected.
NCCEVN_CALL_DISCONNECTED	A DISCONNECT message was received from the network indicating that the remote party has hung up. The event value field contains the reason. The call state changes to disconnected.
NCCEVN_REJECTING_CALL	The application failed to invoke **nccAnswerCall**, **nccAcceptCall**, or **nccRejectCall** within the period specified in the protocol's NCC.START.waitforpctime parameter. The call is automatically rejected and enters the rejecting call state. The event value field contains NCC_REJECT_HOST_TIMEOUT. A tone is played to the network.

nccAutomaticTransfer

Transfers a call on a PBX, Centrex, or Centrex-like line. **nccAutomaticTransfer** executes a blind transfer by performing placement of a second call and completing call transfer. **nccAutomaticTransfer** operates only when the first call handle is in the connected state, or is on hold.

The application determines if a protocol supports this state by examining the NCC_CAP_AUTOMATIC_TRANSFER bit in the capabilitymask returned by **nccQueryCapability**.

The application determines when the call is to be transferred by specifying NCC_TRANSFER_PROCEEDING after the transfer address is dialed.

Associated events	Description
NCCEVN_CALL_HELD	Indicates that the second call (to the transfer address) is in placing call state.
NCCEVN_CALL_RETRIEVED	Indicates that the transfer failed. The event value field contains a NCC_DIS_xxx reason code indicating why the transfer failed.
NCCEVN_CALL_DISCONNECTED	First call is in disconnected call state. Receipt of this event with reason code NCC_DIS_TRANSFER indicates successful completion of the automatic transfer. The application releases this call handle with **nccReleaseCall.**

Other protocol-specific errors or reasons for disconnecting may be reported.

nccBlockCalls

Requests the TCP to block all incoming calls using one of these methods:

- NCC_BLOCK_REJECTALL: The TCP behaves as though the application has responded to each call with **nccRejectCall**. ISDN Software allows two protocol-specific reject modes when NCC_BLOCK_REJECTALL is used. The NCC.X.ADI_ISDN.START_EXT.blockrejectmode parameter determines the mode: 0=reject immediate, 1=reject playing busy tone.
- NCC_BLOCK_OUT_OF_SERVICE: A service request to block calls is sent to the remote end. If this request is confirmed, all subsequent incoming calls are rejected by the switch. This method is supported by the DMS, 4ESS, and E10 variants.

Blocking takes place as soon as there are no calls on the line (the line state is idle). The line state changes to blocking. The line remains in this state and all calls are blocked on the line until **nccUnBlockCalls** is called.

Note: The application should not call **nccUnBlockCalls** until it has received the NCCEVN_CALLS_BLOCKED event.

Associated events	Description
NCCEVN_CALLS_BLOCKED	The request was granted. The line enters blocking line state. The line remains in the blocked state until **nccUnBlockCalls** is called. NCCEVN_CALLS_BLOCKED can also be received as an unsolicited event. See Unsolicited events for more information.
NCCEVN_BLOCK_FAILED	The network failed to respond to the request. The line remains in its current state. The event value field contains a reason code, indicating the reason for the failure. The line remains unblocked.
NCCEVN_CALLS_UNBLOCKED	The switch rejected the request to put the line out of service. The line remains in its current state.

nccDisconnectCall

Disconnects a call that is connected to the network. The function can also be used to abandon outbound call placement.

Associated events	Description
NCCEVN_CALL_DISCONNECTED	Generated after the protocol performs the network procedures for disconnecting the call from the network. The call state changes to disconnected.

nccGetCallStatus

Retrieves the current status of the call (such as caller ID information, if available). For more information, see NCC_CALL_STATUS structure.

There are no associated events with this function.

nccGetExtendedCallStatus

Retrieves protocol-specific status information for a call. For more information, see NCC_ISDN_EXT_CALL_STATUS structure.

There are no associated events with this function.

nccGetLineStatus

Gets current status of line, in NCC_LINE_STATUS.

There are no associated events with this function.

nccHoldCall

Places a call on hold.

nccPlaceCall

Places a call to a specified address, using call placement parameters specified in NCC_ADI_PLACECALL_PARMS. The call enters the outbound initiated state.

Associated events	Description
NCCEVN_PLACING_CALL	Generated after the TCP seizes the trunk and an acknowledgment is received from the network. This implies that glare was resolved and call collision will not occur. The TCP is dialing the digits. The network is sent a SETUP message. The call is now in the placing call state. The event also indicates that a B channel was chosen for the call. This is useful when the TCP is running in non-exclusive mode (see B channel assignment overview). The stream and timeslot for this channel can be determined by calling **nccGetCallStatus.**
NCCEVN_CALL_PROCEEDING	The switch accepted the call setup. The call enters the proceeding state. The NCC.X.ADI_ISDN.START_EXT.startCP parameter determines if call progress analysis is enabled (default) or disabled.
NCCEVN_REMOTE_ALERTING	The network sent an ALERTING message indicating that the remote party is alerted of the call or that a ring tone was detected.
NCCEVN_REMOTE_ANSWERED	The network sent a CONNECT message indicating that the remote party answered.

Associated events	Description
NCCEVN_CALL_CONNECTED	The call satisfies the connectmask requirements set in the NCC_ADI_PLACECALL_PARMS structure. Usually, this is when a CONNECT message is received by the network; however, it could be due to other factors, depending on the connection criteria specified. The event value field indicates the reason. The call is now in the connected call state. The application can play and record voice files, and generate and detect DTMF tones.
NCCEVN_CALL_DISCONNECTED	The call entered the disconnected call state, because: • A release event was received from the network. • A busy, reorder, ring-no-answer, or Special Information Tone (SIT) was detected. • The call fits the disconnection criteria specified in the disconnectmask parameter in the NCC_ADI_PLACECALL_PARMS structure. By default, this is when a DISCONNECT message is received by the network. The event value field contains the condition that fits the disconnection criteria. This is a race condition.
NCCEVN_INCOMING_CALL	A call is arriving on the B channel. Call placement is aborted. This message occurs when a SETUP is received. This is a race condition.
NCCEVN_PROTOCOL_EVENT	An ISDN PROGRESS message was received. The event value field contains 0x9F0E000. The application receives this event if the ISDN_REPORT_PROGRESS bit is set in the NCC.X.ADI_ISDN.START.EXT.ISDNeventmask parameter.

nccRejectCall

Directs the TCP to reject an incoming call using one of the following methods:

- NCC_REJECT_PLAY_BUSY
- NCC_REJECT_PLAY_REORDER
- NCC_REJECT_USER_AUDIO
- NCC_REJECT_PLAY_RINGTONE

Associated events	Description
NCCEVN_REJECTING_CALL	The reject sequence started or the application failed to invoke **nccAnswerCall**, **nccAcceptCall**, or **nccRejectCall** within the period specified in the protocol's NCC.START.waitforpctime parameter. The event value field contains the reject method code, or NCC_REJECT_HOST_TIMEOUT if the host timed out.
NCCEVN_CALL_DISCONNECTED	The remote party hung up. Either a DISCONNECT or a RELEASE message was received from the network. This is a race condition. Optionally, you can invoke **nccDisconnectCall** to cause the TCP to actively clear a rejected call, by sending a DISCONNECT message to the network. No tone is played.

nccReleaseCall

Directs the TCP to release a disconnected call (a call in the disconnected call state). To release a call, the application must first bring it to the disconnected state by invoking **nccDisconnectCall**.

Associated events	Description
NCCEVN_CALL_RELEASED	The call was released. A DISCONNECT message was sent to the network and a RELEASE was received.

nccRetrieveCall

Retrieves a held call.

nccSendCallMessage

Sends ISDN-specific messages to the TCP. For more information, see Sending and receiving ISDN-specific messages.

There are no associated events with this function.

nccSendDigits

Continues the process of sending digits to place an outbound call (for protocols that support overlapped sending of digits). For more information, see Overlapped sending and receiving.

There are no associated events with this function.

nccSendLineMessage

Not supported by ISDN Software.

nccSetBilling

Not supported by ISDN Software.

nccStartProtocol

Starts a TCP.

Associated events	Description
NCCEVN_STARTPROTOCOL_DONE	The function has finished. The event value field indicates if the TCP started successfully: • CTA_REASON_FINISHED: Indicates successful completion. • NCCREASON_OUT_OF_RESOURCES: The application must restart the TCP with the mediamask value in the NCC.ADI_START.CALLCTL structure set to 0.

nccStopProtocol

Stops the protocol and uninitializes the line.

Associated events	Description
NCCEVN_STOPPROTOCOL_DONE	The TCP was halted. The value field is set to CTA_REASON_FINISHED. The line state successfully changed to uninitialized. The line can no longer be used to accept calls, place calls, or both.
NCCEVN_CALL_RELEASED	A call was released before the protocol was stopped.

nccTransferCall

Completes supervised transfer of two calls. When the transfer is completed, the application should invoke **nccReleaseCall** for both call handles to release their resources.

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**.

Associated events	Description
NCCEVN_CALL_DISCONNECTED	Should be returned twice, once for each call. Indicates that the call is disconnected (from the point of view of the application). If the call transfer is successful, the NCC_DIS_TRANSFER reason code is returned with this event. Other protocol-specific reason codes may be reported for failure to complete transfer.

nccUnBlockCalls

Requests the TCP to stop blocking calls.

Associated events	Description
NCCEVN_CALLS_UNBLOCKED	The request to unblock the line is granted. The line state changes to idle.
NCCEVN_UNBLOCK_FAILED	The event value field contains the reason code indicating the reason for the failure. The line remains blocked.
NCCEVN_CALLS_BLOCKED	The switch rejected the request to put the line back into service. The line stays in the blocking state.

Note: The application should not call **nccBlockCalls** again until it has received the NCCEVN_CALLS_UNBLOCKED event.

Unsolicited events

The following table lists the call control-related unsolicited events:

Event	Description
NCCEVN_BILLING_INDICATION	Not supported by ISDN Software.
NCCEVN_CALL_DISCONNECTED	The caller was disconnected from the remote party. The value field contains the reason why the disconnect occurred. Indicates a transition to the disconnected call state.
NCCEVN_CALL_PROCEEDING	The switch accepted the call setup. The receiving side is ringing. This is a secondary and subsequent event generated due to invocation of **nccPlaceCall**. Indicates a transition to the proceeding call state.

Event	Description
NCCEVN_CALL_STATUS_UPDATE	A call status information event was received by NaturalAccess. This can include billing information for outgoing calls. Upon receiving this event, the application can call **nccGetCallStatus** to analyze the relevant fields in the NCC_CALL_STATUS structure.
NCCEVN_CALLID_AVAILABLE	The requested ***callid*** is available.
NCCEVN_CALLS_BLOCKED	This event can be received unsolicited after **nccStartProtocol** returns NCCEVN_STARTPROTOCOL_DONE, if the application blocks a channel with NCC_BLOCK_OUT_OF_SERVICES mode and then restarts the protocol. To restore normal operation, the application should call **nccUnblockCalls** to unblock the channel. The application receives this event after calling **nccBlockCalls**. See Call control functions and solicited events for more information.
NCCEVN_CAPABILITY_UPDATE	Protocol capabilities changed. The application can call **nccQueryCapability** to determine the current set of protocol capabilities.
NCCEVN_EXTENDED_CALL_ STATUS_UPDATE	The call status information. The application can call **nccGetExtendedCallStatus** to determine what changed. The value field contains an indicator showing the kind of information that was received. For more information, see the NCC_ISDN_EXT_CALL_STATUS structure.
NCCEVN_INCOMING_CALL	The TCP has handled the setup of an incoming call, and is now waiting for the application to decide if the call must be answered, accepted, or rejected. The call is now in incoming call state. The application can call **nccGetCallStatus** to retrieve information on the incoming call. The application can then answer, accept, or reject the call as needed within the period of time specified by the NCC.START.waitforPCtime parameter (usually several seconds).
NCCEVN_LINE_IN_SERVICE	The network placed the line in service (the line was out of service). Indicates a transition to the idle line state. The protocol may optionally report more details in the value field.

Event	Description
NCCEVN_LINE_OUT_OF_SERVICE	The called party is blocking the line, or the line and its associated hardware are not configured properly. The TCP will not accept commands until the line comes back into service again. Then the TCP returns to the idle state, and generates an NCCEVN_LINE_IN_SERVICE event. Indicates a transition to the out of service line state. The protocol can optionally report more details of the event in the value field.
NCCEVN_PROTOCOL_ERROR	Received if the application tries to execute functions that are not applicable for the current state, or if a certain function is not supported by a variant. The value field qualifies the event.
NCCEVN_PROTOCOL_EVENT	Received if the network has delivered a message that is not mapped to a standard NCC event. Such messages can be: • An ISDN-specific message. For details, see Sending and receiving ISDN-specific messages. • A PROGRESS message. Delivered only if the NCC.X.ADI_ISDN.START_EXT.ISDNeventmask parameter is set. The value field qualifies the event. No state change takes place.
NCCEVN_RECEIVED_DIGIT	With ISDN Software protocols, you can configure the TCP so that when it receives digits during an incoming call, it sends digits to the application one by one as they arrive (overlap receiving). If the TCP is configured this way, this event indicates that a digit has arrived. The event value field is set to the value of the digit (a char). The call state changes to receiving digits, if it is not already in that state.
NCCEVN_REJECTING_CALL	(Solicited) **nccRejectCall** was invoked, and the call is now rejected. (Unsolicited) The application failed to answer, accept, or reject the call in a timely fashion. The call is automatically rejected. In both cases, expect NCCEVN_CALL_DISCONNECTED. The value field contains a qualifier for why call a is rejected. Indicates a transition to the rejecting call state.

Event	Description
NCCEVN_SEIZURE_DETECTED	The line was seized by an external entity for an incoming call. It is the beginning of the call setup process for an incoming call. Indicates a transition to the seizure call state.

Retrieving call information

Invoke the following functions to obtain status information on a call:

Function	Description
nccGetCallStatus	NCC_CALL_STATUS structure containing information about the call, such as the DID and ANI information, or the current call state.
nccGetExtendedCallStatus	NCC_ISDN_EXT_CALL_STATUS structure containing protocol-specific information about the call.

You can invoke either of these synchronous functions in any call state, as long as the line is in active line state.

NCC_CALL_STATUS structure

nccGetCallStatus returns the NCC_CALL_STATUS structure:

```
#define NCC_MAX_DIGITS 31
#define NCC_MAX_CALLING_NAME 63
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+1]; /* 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;
```

The NCC_CALL_STATUS structure contains the following fields:

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

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 called party address. For ISDN Software, the digits are formatted as follows: $\boldsymbol{d}_1 \ldots \boldsymbol{d}_n$ * $\boldsymbol{t}_1 \ldots \boldsymbol{t}_n$ ...where: • $\boldsymbol{d}_1 \ldots \boldsymbol{d}_n$ are the digits, and • $\boldsymbol{t}_1 \ldots \boldsymbol{t}_n$ are the subaddress digits, if available. If no subaddress is available, the * and ***t*** digits do not appear in the string.
callingaddr	For inbound calls, the calling party address - the automatic number identification (ANI) digits. For ISDN Software, the digits are formatted as follows: $\boldsymbol{a}_1 \ldots \boldsymbol{a}_n$ * $\boldsymbol{s}_1 \ldots \boldsymbol{s}_n$ ...where: • $\boldsymbol{a}_1 \ldots \boldsymbol{a}_n$ are the ANI digits, and • $\boldsymbol{s}_1 \ldots \boldsymbol{s}_n$ are the subaddress digits, if available. If no subaddress is available, the * and s digits do not appear in the string.
callingname	(Inbound calls) The name information of the caller, if provided.

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

Because the values of these fields depend on the information associated with the incoming call, and on the protocol used to set up the call, not all fields are necessarily filled during call set up.

Each ISDN variant fills different fields in the NCC_CALL_STATUS structure at different times. The following table shows which fields are filled by which variant. Each field may be filled at any time with the exception of the callingaddr and pendingcmnd fields. The callingaddr and pendingcmnd fields may be filled at the beginning of the call.

Field	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T1607
size	x	x	x	x	x	x	x	x	x	x	x	x	x	x
state	x	x	x	x	x	x	x	x	x	x	x	x	x	x
calledaddr	x	x	x	x	x	x	x	x	x	x	x	x	x	x
callingaddr	x	x	x	x	x	x	x	x	x	x	x	x	x	x
callingname	x											x		

Field	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T16007
pendingcmd	x	x	x	x	x	x	x	x	x	x	x	x	x	x
held														
direction	x	x	x	x	x	x	x	x	x	x	x	x	x	x
linehd	x	x	x	x	x	x	x	x	x	x	x	x	x	x

NCC_ISDN_EXT_CALL_STATUS structure

nccGetExtendedCallStatus returns the NCC_ISDN_EXT_CALL_STATUS structure:

```
typedef struct
{                                   /*                                      */
    DWORD size ;                        /* size of this structure               */
    INT32 reason;                       /* reason of going back to IDLE state   */
    DWORD stream;                       /* mvip stream number                   */
    DWORD timeslot;                     /* mvip timeslot                        */

    WORD  callid[NCC_ISDN_CALLID_LEN];  /* call identifier                      */
    WORD  callreference;                /* Q.931 call reference                 */

    DWORD chargingvalue;                /* charging value                       */
    char  chargingmulti;                /* charging multiplier                  */
    char  chargingtype;                 /* charging type                        */
    char  chargingperiod;               /* charging period                      */

    char  callednumplan;                /* Q.931 numbering plan ID if supported */
    char  callednumtype;                /* Q.931 number type if supported       */
    char  callingnumplan;               /* Q.931 numbering plan ID if supported */
    char  callingnumtype;               /* number type if supported             */
    char  callingpres;                  /* caller ID presentation indicator     */
    char  callingscreen;                /* Q.931 ANI screening indicator        */

    char  progressdescr;                /* progress descriptor                  */

    char  releasecause;                 /* cause for call release (or PROGRESS) */

    char  calledsubaddr[33];            /* Called sub-address                   */
    char  calledsubaddrtype;            /* Called sub-address type              */
    char  calledsubaddroddeven;         /* Called sub-address odd-even indicator*/
    char  callingsubaddr[33];           /* Calling sub-address                  */
    char  callingsubaddrtype;           /* Calling sub-address type             */
    char  callingsubaddroddeven;        /* Calling sub-address odd-even indicator */


    char  redirectingaddr[33];          /* redirecting number                   */
    char  redirectingplan;              /* Q.931 numbering plan ID if supported */
    char  redirectingtype;              /* Q.931 number type if supported       */
    char  redirectingpres;              /* redirecting number pres. indicator   */
    char  redirectingscreen;            /* Q.931 redirecting number screen ind. */
    char  redirectingreason;            /* Q.931 reason for redirection         */

    char  redirectionaddr[33];          /* redirection number                   */
    char  redirectionplan;              /* Q.931 numbering plan ID if supported */
    char  redirectiontype;              /* Q.931 number type if supported       */
```

```
    char  redirectionpres;              /* redirection number pres. indicator     */
    char  redirectionscreen;            /* Q.931 redirection number screen ind.   */
    char  redirectionreason;            /* Q.931 reason for redirection           */

    char  originalcalledaddr[33];       /* original called number                 */
    char  origcalledplan;               /* Q.931 numbering plan ID if supported   */
    char  origcalledtype;               /* Q.931 number type if supported         */
    char  origcalledpres;               /* original called number pres. indicator */
    char  origcalledscreen;             /* Q.931 orig called number screen ind.   */
    char  origcalledreason;             /* Q.931 reason for redirection           */
    char  origcalledcount;              /* Q.931 redirection counter              */
    char  origcalledcfnr;               /* Q.931 call forward no response indic.  */

    char  UUI[NCC_ISDN_MAX_UUI + 1];    /*   user to user information             */

    char  connectedname[32];            /* connected name                         */
    char  connectedaddr[33];            /* connected number                       */
    char  connectedplan;                /* Q.931 numbering plan ID if supported   */
    char  connectedtype;                /* Q.931 number type if supported         */
    char  connectedpres;                /* connected number presentation indicator*/
    char  connectedscreen;              /* Q.931 connected number screening ind.  */

    char  origlineinfo;                 /* originating line information (ANI II)  */
                                        /* or Calling Party Category              */
    char  char  national_cpc;           /* National Calling Party Category        */

    char  char  nsf_present;            /* Network Specific Facility usage flag   */
    char  char  nsf_service_feature;    /* service or feature selector            */
    char  char  nsf_facility_coding;    /* nsf coding                             */
    char  char  nsf_param_fld;          /* nsf parametrized facility coding value */

    char  char  service;                /* call service                           */
    char  char  pad1[5];

} NCC_ISDN_EXT_CALL_STATUS;
```

The following table describes the fields in the NCC_ISDN_EXT_CALL_STATUS structure. Not all fields are supported by all variants.

Except as marked, all parameter values are set to 0 (zero) by default. Possible values for the fields in this structure are defined in *isdnval.h*.

Because the values of these fields depend on the information associated with the incoming call, and on the protocol used to set up the call, not all fields are necessarily filled during call set up.

Field	Description
size	Number of bytes written at the address pointed to by the ***status*** argument in **nccGetExtendedCallStatus**.
reason	Reason for the last disconnect. reason is 0 if the application initiated the disconnect. Otherwise, reason is the NCC disconnect value received in the NCCEVN_CALL_DISCONNECTED event.
stream	This field and timeslot together indicate the address of the B channel. Use if the TCP is in non-exclusive mode. (See B channel assignment overview.)
timeslot	This field and stream together indicate the address of the B channel. Use if the TCP is in non-exclusive mode. (See B channel assignment overview.)

Field	Description
callid	Call identifier for transfer.
callreference	Q.931 call reference associated with the current call.
chargingvalue	Charging value (number of units).
chargingmulti	Charging multiplier.
chargingtype	Charging type.
chargingperiod	Charging period (amount of time).
callednumplan	Q.931 numbering plan of called address.
callednumtype	Q.931 numbering type of called address.
callingnumplan	Q.931 numbering plan of calling address.
callingnumtype	Q.931 numbering type of calling address.
callingpres	Q.931 presentation indicator for calling address.
callingscreen	Q.931 screening indicator for calling address.
progressdescr	Q.931 progress description in progress information element.
releasecause	Q.931 cause for call release.
calledsubaddr[33]	Called subaddress.
calledsubaddrtype	Called subaddress type.
calledsubaddroddeven	Called subaddress odd/even indicator.
callingsubaddr[33]	Calling subaddress.
callingsubaddrtype	Calling subaddress type.
callingsubaddroddeven	Calling subaddress odd/even indicator.
redirectingaddr[33]	Redirecting address.
redirectingplan	Q.931 numbering plan of redirecting address.
redirectingtype	Q.931 numbering type of redirecting address.

Field	Description
redirectingpres	Q.931 presentation indicator for redirecting address.
redirectingscreen	Q.931 screening indicator for redirecting address.
redirectingreason	Q.931 reason for redirection.
redirectionaddr[33]	Redirection address.
redirectionplan	Q.931 numbering plan of redirection address.
redirectiontype	Q.931 numbering type of redirection address.
redirectionpres	Q.931 presentation indicator for redirection address.
redirectionscreen	Q.931 screening indicator for redirection address.
redirectionreason	Q.931 reason for redirection.
originalcalledaddr[33]	Original called number (OCN).
origcalledplan	Q.931 OCN numbering plan.
origcalledtype	Q.931 OCN numbering type.
origcalledpres	Q.931 OCN presentation indicator.
origcalledscreen	Q.931 OCN screen indicator.
origcalledreason	Q.931 OCN reason for redirection.
origcalledcount	Q.931 OCN redirection counter.
origcalledcfnr	Q.931 OCN call forward no response indicator.
UUI	User-to-user information (up to 132 characters).
connectedname[32]	Name of the connected party.
connectedaddr[33]	Number of the connected party.
connectedplan	Q.931 numbering plan of connected number.
connectedtype	Q.931 numbering type of connected number.
connectedpres	Q.931 presentation indicator for connected number.

Field	Description
connectedscreen	Q.931 screen indicator for connected number.
origlineinfo	Originating line information. The default value for origlineinfo is 0xFF. This value indicates that originating line information is not available.
national_cpc	National calling party category.
nsf_present	Availability of Network Specific Facility (NSF) information. 0 - Not available 1 - Available (see other nsf_xxx fields).
nsf_service_feature	Service or feature is set in the coding field of NSF.
nsf_facility_coding	NSF service or feature ID.
nsf_param_fld	NSF parameterized facility coding value.
service	xxx_SERVICE constants from *isdnval.h*.

Each ISDN variant fills different fields in the NCC_ISDN_EXT_CALL_STATUS structure at different times. An x in the column of the following table shows which fields are filled by which variant. Unless otherwise indicated, each field can be filled at any time.

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T1607
size		x	x	x	x	x	x	x	x	x	x	x	x	x	x
reason	When the call is released.	x	x	x	x	x	x	x	x	x	x	x	x		x
stream		x	x	x	x	x	x	x	x	x	x	x	x	x	x
timeslot		x	x	x	x	x	x	x	x	x	x	x	x	x	x
callid		x	x									x	x	x	
callreference	At the beginning of the call or when the call is alerting.		x												

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T16007
chargingvalue	When the call is in the connected state or when the call is released.										x				
chargingmulti	When the call is in the connected state or when the call is released.										x				
chargingtype	When the call is in the connected state or when the call is released.										x				
chargingperiod	When the call is in the connected state or when the call is released.										x				
callednumplan	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
callednumtype	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
callingnumplan	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
callingnumtype	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
callingpres	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
callingscreen	At the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T16007
progressdescr	At the beginning of the call, when the call is alerting, or when the call is released. For Q.SIG, the field can be filled at the beginning of the call.	x	x	x	x	x	x	x	x	x	x	x	x		x
releasecause	When the call is released or a PROGRESS message is received.	x	x	x	x	x	x	x	x	x	x	x	x		x
calledsubaddr[33]	At the beginning of the call.	x	x			x	x	x	x		x	x	x		
calledsubaddrtype	At the beginning of the call.						x					x	x		
calledsubaddroddeven	At the beginning of the call.											x	x		
callingsubaddr[33]	At the beginning of the call.	x	x			x	x	x	x		x	x	x		
callingsubaddrtype	At the beginning of the call.						x					x	x		
callingsubaddroddeven	At the beginning of the call.											x	x		
redirectingaddr[33]	At the beginning of the call or when a call is redirected.	x	x	x	x	x					x				

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T1607
redirectingplan	At the beginning of the call or when a call is redirected.	x	x	x	x	x									
redirectingtype	At the beginning of the call or when a call is redirected.	x	x	x	x	x					x				
redirectingpres	At the beginning of the call or when a call is redirected.	x	x	x	x	x					x				
redirectingscreen	At the beginning of the call or when a call is redirected.	x	x	x	x	x					x				
redirectingreason	At the beginning of the call or when a call is redirected.	x	x	x	x	x					x				
redirectionaddr[33]	At the beginning of the call or when a call is redirected.	x													
redirectionplan	At the beginning of the call or when a call is redirected.	x													
redirectiontype	At the beginning of the call or when a call is redirected.	x													
redirectionpres	At the beginning of the call or when a call is redirected.	x													

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T16007
redirectionscreen	At the beginning of the call or when a call is redirected.	x													
redirectionreason	At the beginning of the call or when a call is redirected.	x													
originalcalledaddr[33]	DMS: At the beginning of the call. EUR: Whenever keypad information is received in the connected state.	x										x			
origcalledplan	At the beginning of the call.	x													
origcalledtype	At the beginning of the call.	x													
origcalledpres	At the beginning of the call.	x													
origcalledscreen	At the beginning of the call.	x													
origcalledreason	At the beginning of the call.	x													
origcalledcount	At the beginning of the call.	x													
origcalledcfnr	At the beginning of the call.	x													

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T16007
UUI	At the beginning of the call, when the call is alerting, or when the call is released. For 5ESS and HKT, the field may be filled at the beginning of the call or when the call is released. For Q.SIG, the field may be filled when the call is alerting or when the call is released.		x	x	x	x		x	x	x	x	x	x		
connectedname[32]	When the call is in the connected state.	x													
connectedaddr[33]	When the call is in the connected state.	x		x	x								x		x
connectedplan	When the call is in the connected state.	x		x	x								x		x
connectedtype	When the call is in the connected state.	x		x	x								x		x
connectedpres	When the call is in the connected state.	x		x	x								x		x
connectedscreen	When the call is in the connected state.	x		x	x								x		x

Field	This field can be filled...	DMS	NI2	4ESS	5ESS	NTT	AUS	HKT	KOR	TWN	VN6	EUR	QSIG	DPNSS	T1607
origlineinfo	At the beginning of the call.	x		x								x			
national_cpc	At the beginning of the call.											x			
nsf_present	At the beginning of the call.			x											
nsf_service_feature	At the beginning of the call.			x											
nsf_facility_coding	At the beginning of the call.			x											
nsf_param_fld	At the beginning of the call.			x											
service	At the beginning of the call.			x											

Some fields are static during the call. Whenever any of these fields changes, the application receives NCCEVN_EXTENDED_CALL_STATUS_UPDATE. The value field returns with this event containing a mask indicating the fields that changed:

Mask bit	Description
NCC_X_STATUS_INFO_UUI	UUI field was changed.
NCC_X_STATUS_INFO_PROGRESSDESCR	progressdescr field was changed.
NCC_X_STATUS_INFO_CHARGE	One or more of the charging***xxx*** fields was changed.
NCC_X_STATUS_INFO_CONN_NAME	Name of the connected party was received.
NCC_X_STATUS_INFO_CAUSE	releasecause field was updated.
NCC_X_STATUS_INFO_CONN_NUMBER	Connected number was received.
NCC_X_STATUS_INFO_REDIRECTING	Redirecting number was received.
NCC_X_STATUS_INFO_REDIRECTION	Redirection number was received.

Mask bit	Description
NCC_X_STATUS_INFO_KEYPAD	Keypad information in origcalledaddr field has changed.
NCC_X_STATUS_INFO_CALL_RELEASED	Network has released the disconnected call.

Refer to the file *nms\include\isdnval.h* that ships with the product. This file contains the values for many of the fields found in the extended parameter structures and in the NCC_ISDN_EXT_CALL_STATUS structure.

Digit strings in outbound calls

Calling address (ANI) and DNIS information is passed in two separate digit strings in **nccPlaceCall** invocations:

- The called party information is passed in the ***calledaddr*** argument. The ISDN Software TCP expects this digit string to be formatted as:

  $d_1 \ldots d_n$ * $t_1 \ldots t_n$

  ...where:

  $d_1 \ldots d_n$ are the digits to dial.

  $t_1 \ldots t_n$ are the subaddress digits to dial, if needed. If no subaddress is used, omit the * and ***t*** digits from the string.

- The calling party information is passed in the ***callingaddr*** argument. The ISDN Software TCP expects this digit string to be formatted as:

  $a_1 \ldots a_n$ * $s_1 \ldots s_n$

  ...where:

  $a_1 \ldots a_n$ are the address of the calling party.

  $s_1 \ldots s_n$ are the subaddress digits of the calling party, if used. If no subaddress is used, omit the * and ***s*** digits from the string.

Overlapped sending and receiving

Some variants of the ISDN Software protocol support overlapped sending and overlapped receiving: the ability to send or receive address digits one at a time.

Capability mask lists the capabilities for each variant. If OVERLAPPED_SENDING is set, the variant supports overlapped sending. The application determines whether a variant supports this capability by invoking **nccQueryCapability**. The NCC_CAP_OVERLAPPED_SENDING bit in the capability mask returned by this function indicates if the variant supports this capability.

Note: DPNSS supports overlapped sending and overlapped receiving.

Setting up overlapped sending and receiving

If the ISDN variant supports overlapped sending and receiving, to use the features the application must change bits in the behavior parameter structures when starting the ISDN Software protocol stack.

Feature	Description
Performing overlapped sending	The application much change a bit in the out_calls_behaviour parameter in the ISDN_PROTOCOL_PARMS_ CHANNELIZED structure when starting the ISDN Software protocol stack. The application must enable the CC_USER_SENDING_COMPLETE (0x0002) bit to prevent the stack from automatically setting it in the SETUP message. Command line switch -O (capital O) in the ISDN daemon *isdncta* sets this bit.
Performing overlapped receiving	The application much change a bit in the in_calls_behaviour parameter in the ISDN_PROTOCOL_PARMS_CHANNELIZED structure when starting the ISDN Software protocol stack. The bit CC_TRANSPARENT_OVERLAP_RCV (0x0080) should be enabled. Command line switch (-I) in the ISDN daemon *isdncta* sets this bit. In addition, the overlappedreceiving parameter must be set in the NCC_START_PARMS structure (invoked when **nccStartProtocol** is called) to inform the TCP that overlapped receiving mode is requested by the application. By default, this parameter is set. The NCCEVN_RECEIVED_DIGIT event (indicating that a digit has arrived) is generated only if the NCC.START.overlappedreceiving parameter is set. See the *Natural Call Control Service Developer's Reference Manual.*

For more information about these parameter structures, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

Performing overlapped sending and receiving

Once overlapped sending and receiving are set up, you can perform overlapped transfers.

Feature	Description

Performing overlapped sending	To perform overlapped sending, the application invokes **nccPlaceCall**, passing the first sequence of digits in the ***calledaddr*** argument. If more digits will follow, the digit string must end with a ~ character.
	After the application receives NCCEVN_PLACING_CALL, indicating that it has reached the placing call state, it invokes **nccSendDigits** to send additional segments of the digit string. If a digit string sent by an invocation of **nccPlaceCall** or **nccSendDigits** is not the final one, the final character in the string must be a ~. The final digit string must not include the final ~ character.
	For example:
	`nccPlaceCall 72~`
	`ccSendDigits 57924~`
	`ccSendDigits 5~`
	`ccSendDigits 28`
	Final digit string:
	`7257924528`
	nccSendDigits can be called several times until all digits are sent, or the switch sends NCCEVN_CALL_PROCEEDING indicating that it has accepted the call setup. No digits can be sent after this point.

Performing overlapped receiving	An incoming call begins in the seizure call state. When the first digit arrives, the application receives an NCCEVN_RECEIVED_DIGIT event. This places the call in receiving digits call state. Each subsequent NCCEVN_RECEIVED_DIGIT event indicates that another digit has been received. The call remains in the receiving digits call state until all digits are received, or the application calls an appropriate call control function (for example, **nccAcceptCall** or **nccAnswerCall**). As each digit is received, it is added to the calledaddr field in the NCC_CALL_STATUS structure. NCCEVN_INCOMING_CALL indicates that all digits have arrived. If call collision (glare) occurs directly after the application has issued a command, NCCEVN_RECEIVED_DIGIT or NCCEVN_INCOMING_CALL can be returned before the confirmation event for the command (for example between **nccAcceptCall** and NCCEVN_ACCEPTING_CALL). The event changes the call state from receiving digits to a new state.

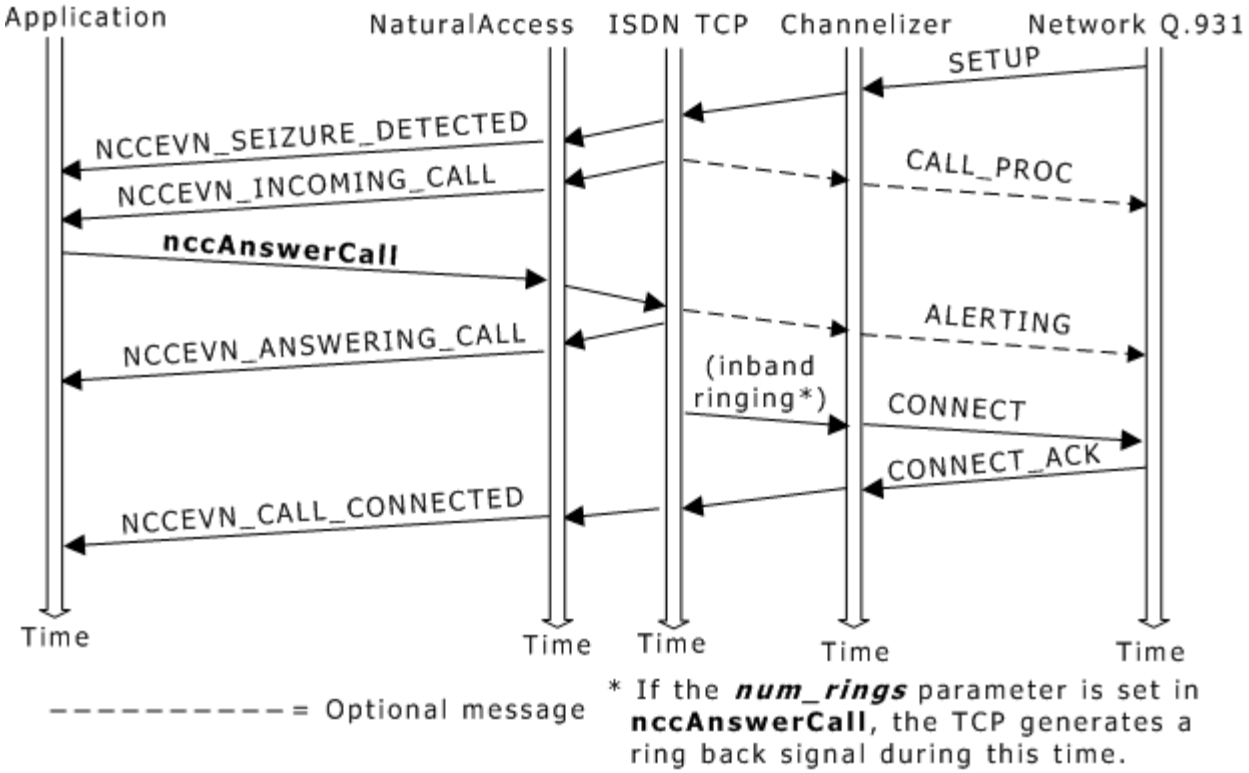
Sequence diagrams

This topic shows the normal exchange of commands and events between the network, the channelizer, the TCP, NaturalAccess, and the application during various NCC service call control operations. The following function sequences are presented:

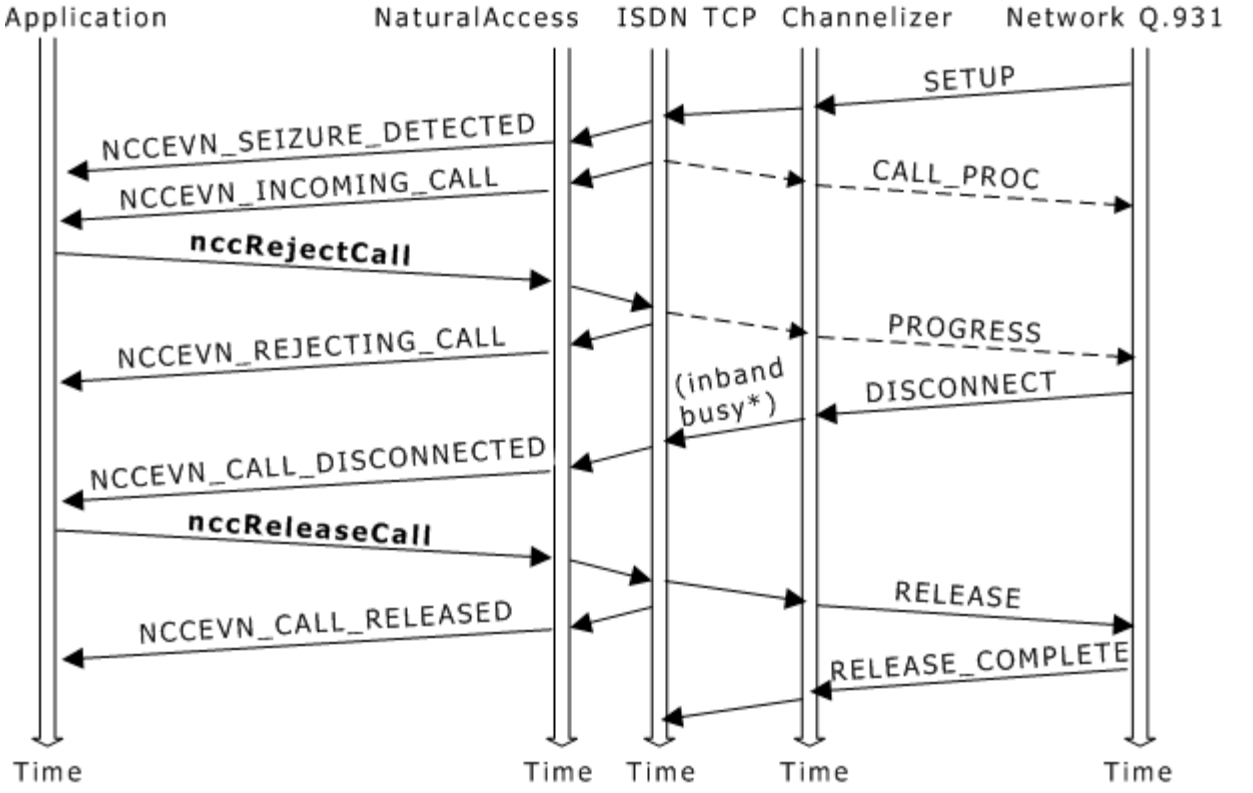
- Inbound calls
- Outbound calls
- Overlapped sending and receiving sequence diagrams
- Disconnecting and releasing

Inbound calls

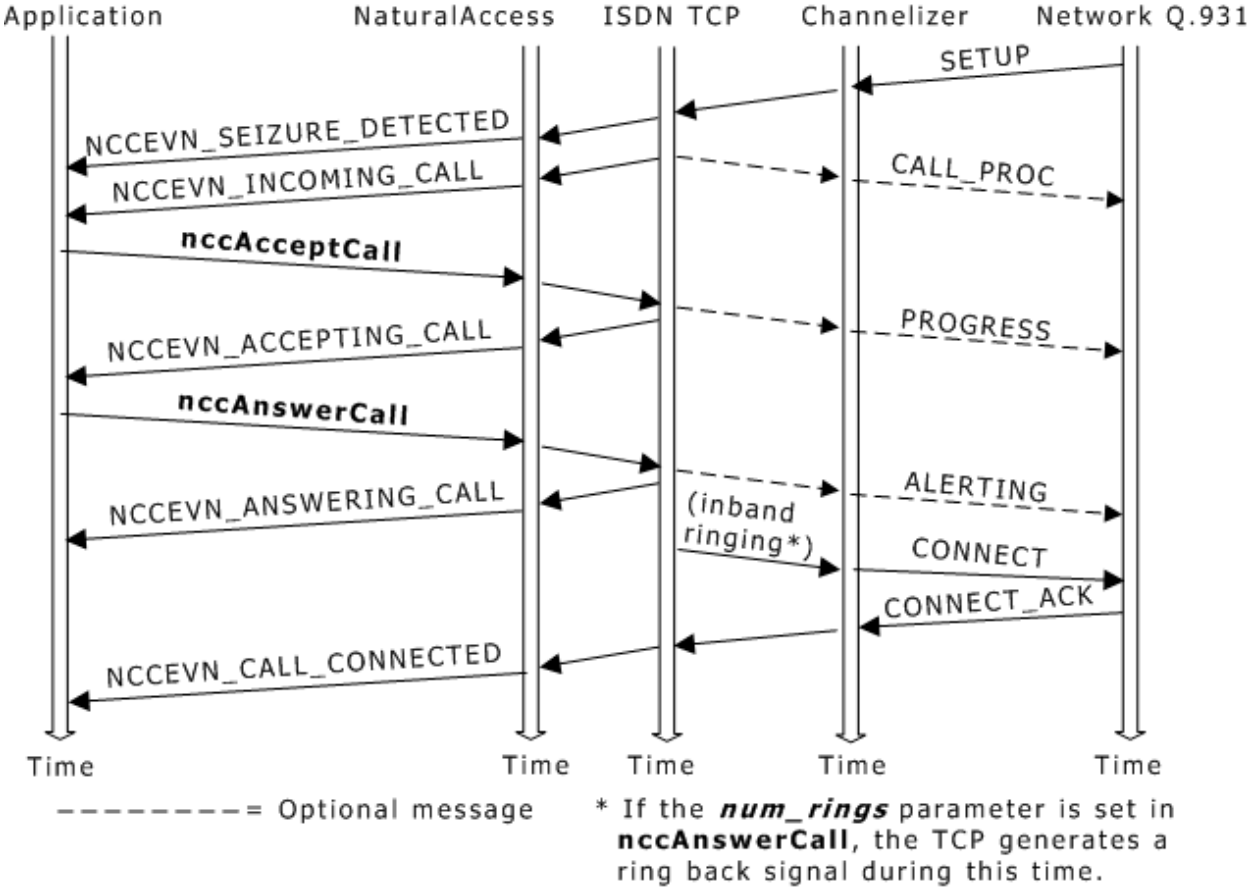
The following illustration shows the function sequence for answering an inbound call:



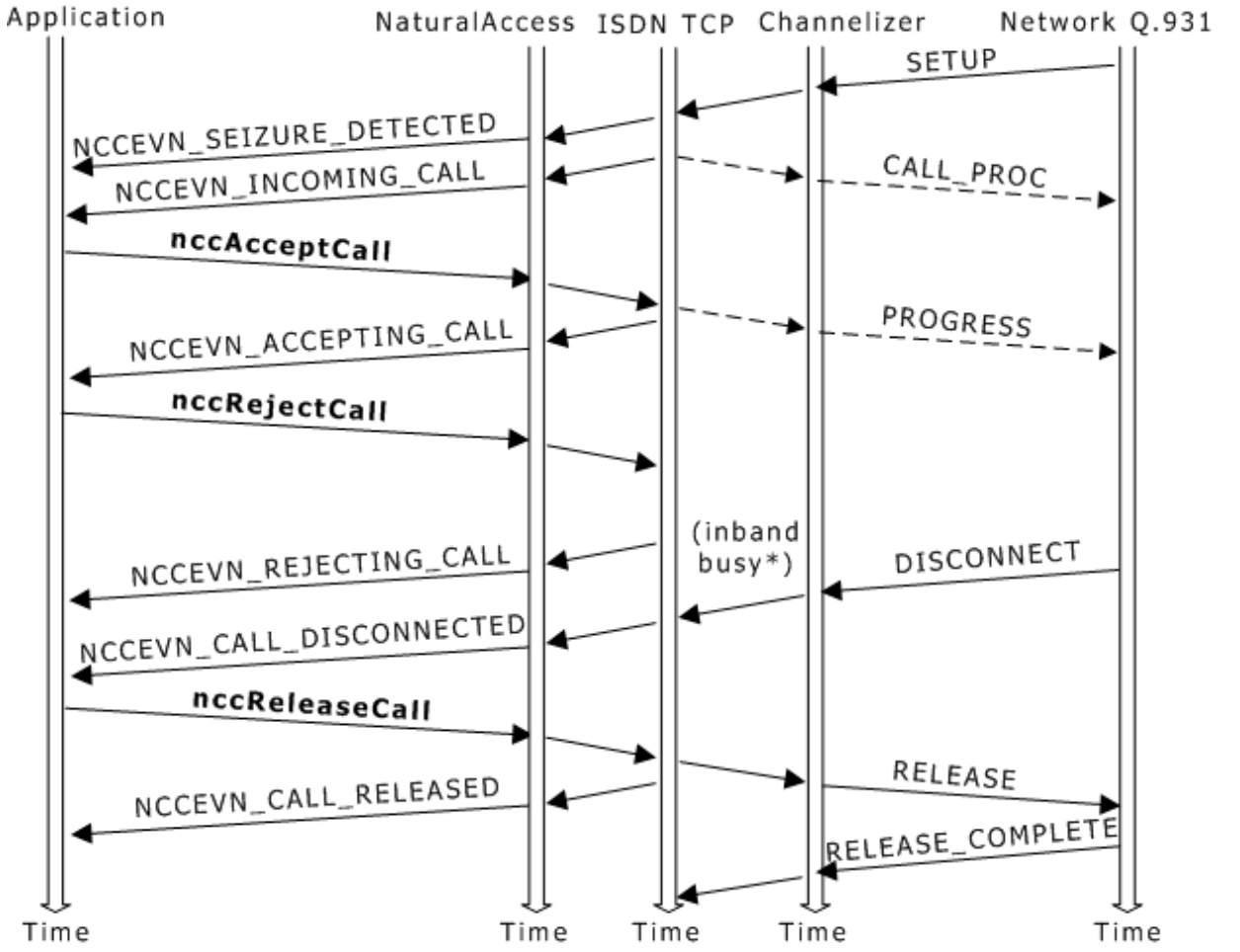
The following illustration shows the function sequence for rejecting an inbound call:



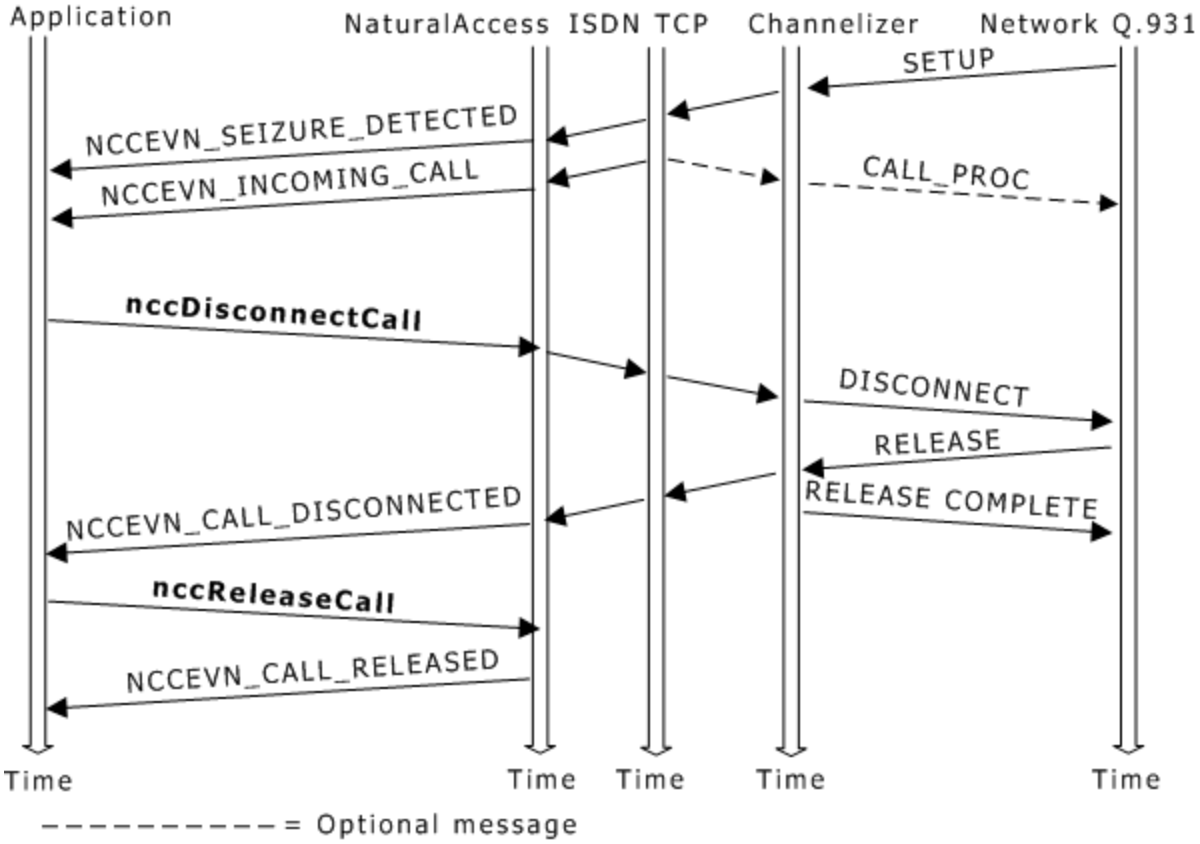
The following illustration shows the function sequence for accepting an inbound call and then answering it:



The following illustration shows the function sequence for accepting an inbound call and then rejecting it:

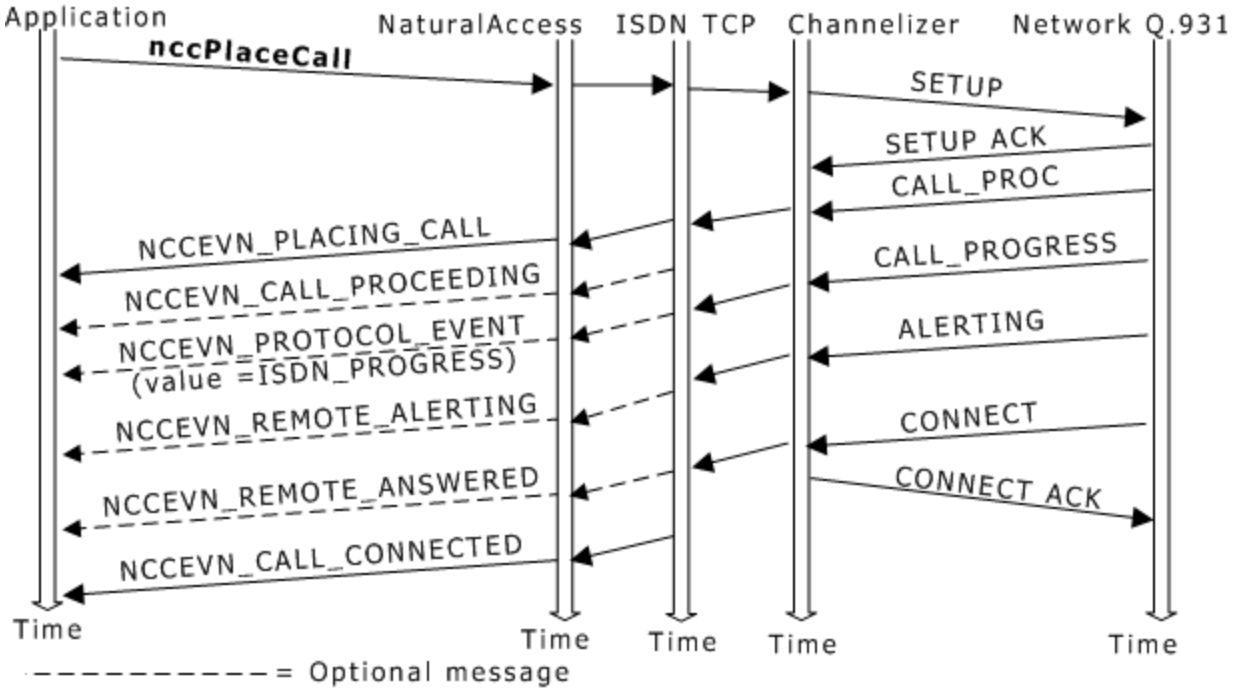


The following illustration shows the function sequence for immediately rejecting an incoming call:



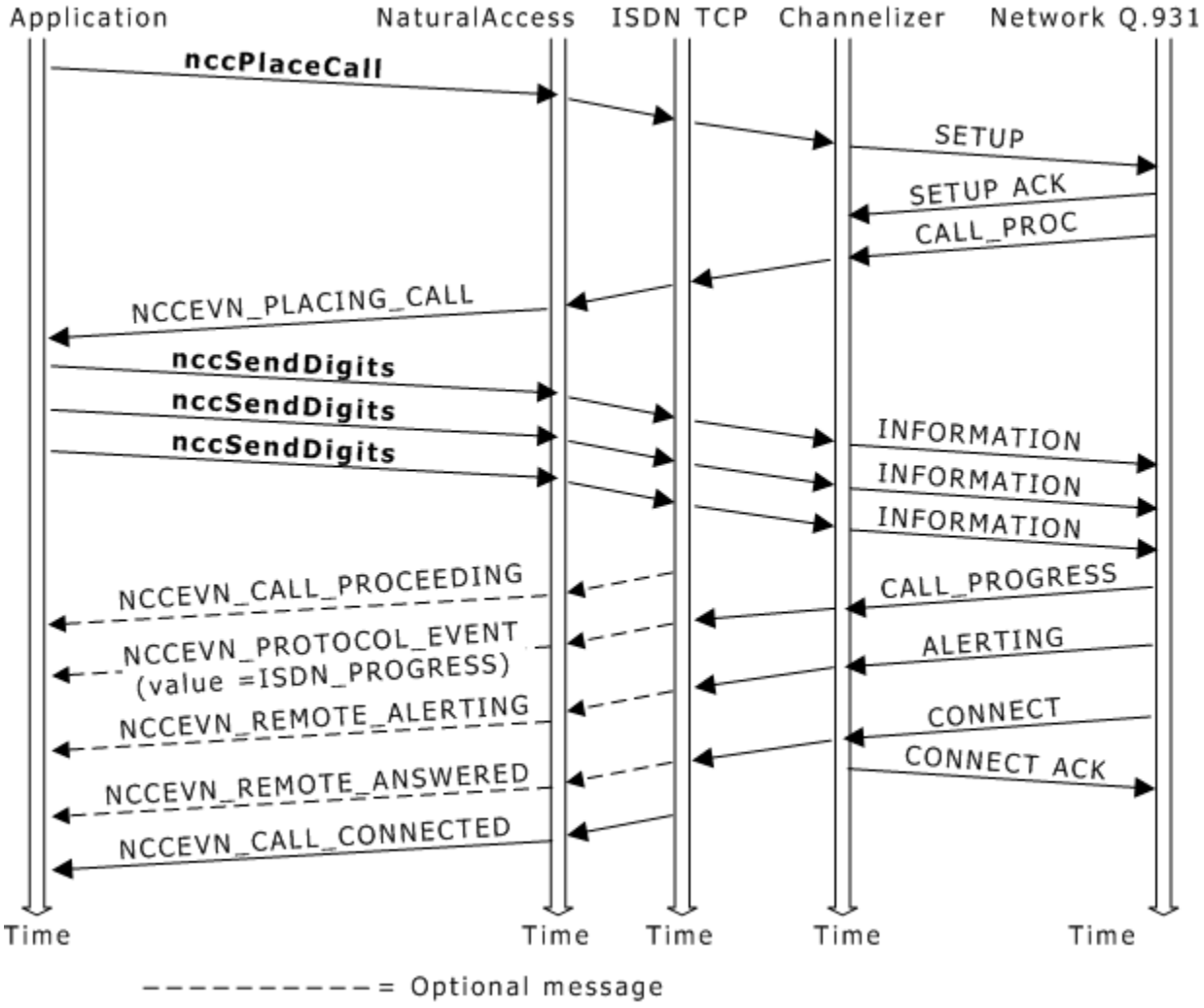
Outbound calls

The following illustration shows the function sequence for placing an outbound call. Depending upon the switch variant, some of the intermediate messages may not occur, or may occur in a different order than shown here.

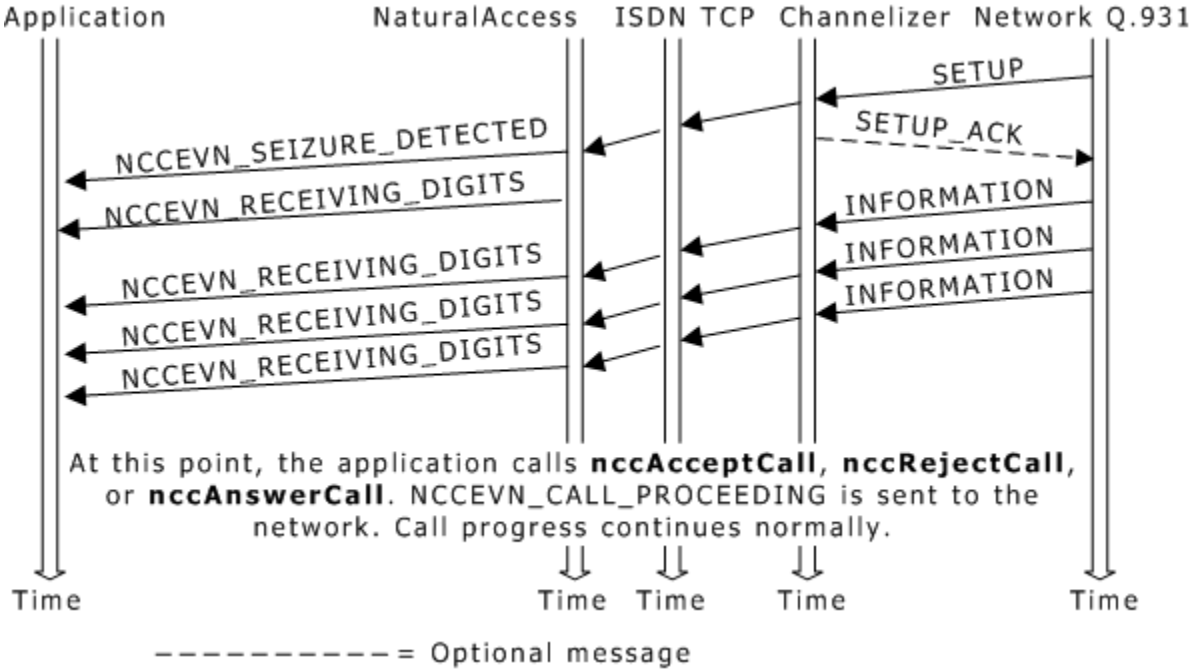


Overlapped sending and receiving sequence diagrams

The following illustration shows the function sequence for placing a call with overlapped sending of digits:

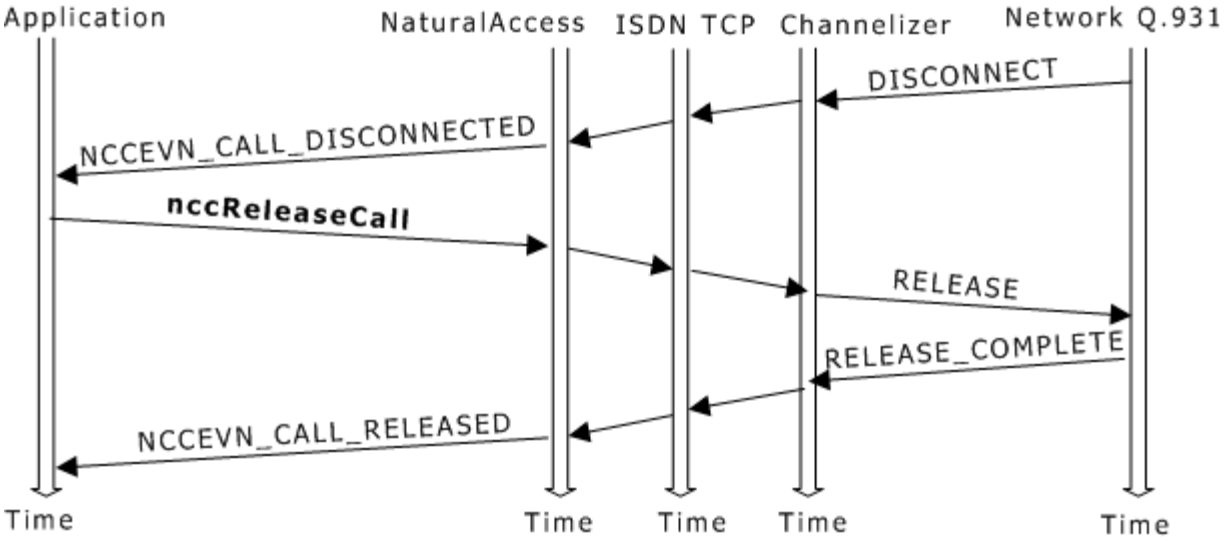


The following illustration shows events returned while handling an inbound call with overlapped receiving of digits:

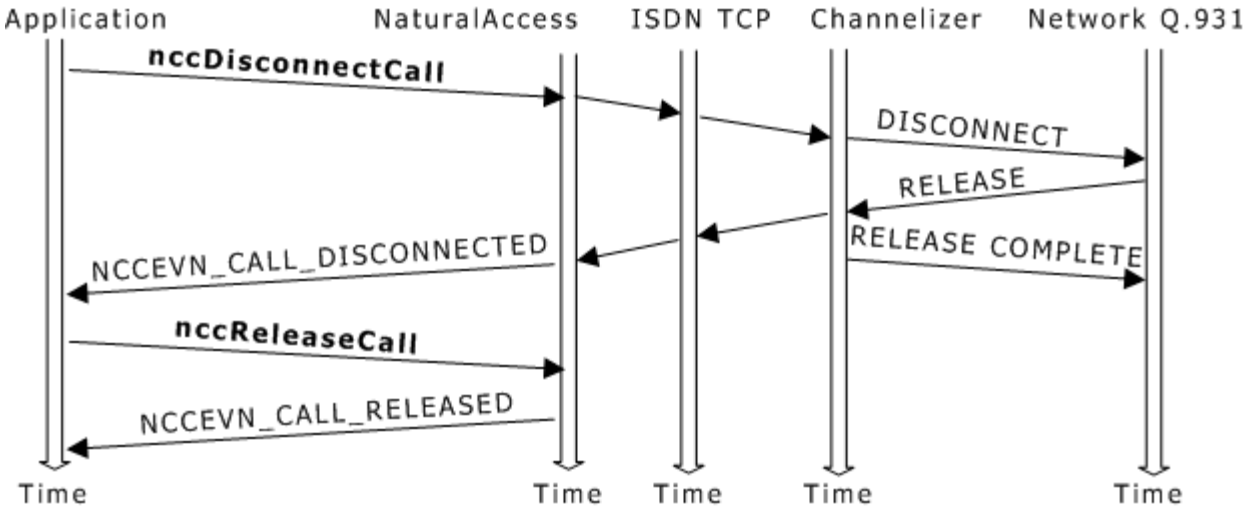


Disconnecting and releasing

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



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



Capability mask

An application can call **nccQueryCapability** to determine the capabilities of a protocol. **nccQueryCapability** returns a capabilitymask. For more information, see the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

Capabilities differ slightly from variant to variant. The capabilitymask also varies depending upon whether the stack is the NT or TE side. The following table shows which capabilities the capabilitymask supports for each variant.

For this table, NT = NT side only, TE = TE side only, and x = both NT and TE sides.

Capability	DMS	NI2	4ESS	5ESS	AUS1	HKT	KOR	NTT	TWN	ETSI	VN6	QSIG	DPNSS	T16607
CALLER_ID	x	x	x	x	x	x	x	x	x	x	x	x		x
MEDIA_IN_SETUP	x	x	x	x	x	x	x	x	x	x	x	x		x
DISCONNECT_IN_ANY_STATE	x	x	x	x	x	x	x	x	x	x	x	x		x
HOLD_IN_ANY_STATE														
SEND_LINE_MESSAGE														
SEND_CALL_MESSAGE	x	x	x							TE		TE		
EXTENDED_CALL_STATUS	x	x	x	x	x	x	x	x	x	x	x	x		x
NCC_CAP_TWO CHANNEL_TRANSFER	TE	TE												
AUTOMATIC_TRANSFER			TE											
SUPERVISED_TRANSFER														
HOLD_CALL														
OVERLAPPED_SENDING							x		x	x	x	x		x
SET_BILLING														
ACCEPT_CALL	x	x	x	x	x	x	x	x	x	x	x	x		x

Note: When the stack is down, only EXTENDED_CALL_STATUS capabilities are enabled.

6. ISDN-specific messages

Sending and receiving ISDN-specific messages

Some variants of the ISDN Software protocol support sending and receiving ISDN-specific messages for which there are no NCC API equivalents. ISDN Software currently supports the following ISDN-specific features:

This feature...	Allows an application to...
Transparent message sending and receiving	Send user-defined information elements (IEs) to the stack, and receive FACILITY messages that are not recognized by the stack.
Request call ID for call transfer	Request an identifier for one of two calls transferred together. This feature is useful when both calls are inbound.

Sending ISDN-specific messages

Not all variants support the sending and receiving of ISDN-specific messages. The capability mask lists the capabilities for each variant. The application determines whether a variant supports sending and receiving of ISDN messages by invoking **nccQueryCapability**. The NCC_CAP_SEND_CALL_MESSAGE bit in the capability mask returned by this function indicates if the variant supports this capability.

To send ISDN-specific messages, use **nccSendCallMessage**. **nccSendCallMessage** takes the following arguments:

Argument	Description
callhd	Call handle to which message should be sent.
message	Pointer to buffer containing the message structure.
size	Size of the buffer.

All ISDN-specific structures and macros associated with NCC are defined in *nccxisdn.h*. Each of the structures in this file contains:

- A header, identifying the id and the type of the message.
- (Optional) The rest of the structure specific to the message.

The header structure is:

```
typedef struct
{
    WORD message_id;              /* Message id                */
    WORD message_type;            /* Message type              */
} NCC_ISDN_SEND_CALL_MESSAGE;
```

message_id field values

The following table lists possible values for the message_id field:

message_id value	Description
NCC_ISDN_TRANSPARENT_BUFFER	Transparent message sending operation.
NCC_ISDN_TRANSFER_CALLID_RQ	Request a call identifier for a call to be transferred.

message_type field values

The following table lists possible values that can be returned in the message_type field:

message_type value	Description
NCC_ISDN_RETURN_REJECT	Rejection of a request to invoke the operation.
NCC_ISDN_RETURN_RESULT	Successful completion of the operation.
NCC_ISDN_RETURN_ERROR	Unsuccessful completion of the operation.

To invoke an operation, use any of NCC_ISDN_***xxx***_INVOKE structures, where ***xxx*** is the ID of the message. The structure to use depends upon what operation the application is to perform. For example, the structure used to invoke TBCT is NCC_ISDN_TBCT_INVOKE.

Receiving ISDN-specific events

When an ISDN-specific event is presented to the application, the application receives NCCEVN_PROTOCOL_EVENT. The event value field contains ISDN_CALL_MESSAGE_SENT, and the buffer field (returned with the event ID in the CTA_EVENT structure) provides a pointer to a structure containing the result. This structure is NCC_ISDN_***xxx_yyy***, where ***xxx*** is the id of the ISDN-specific feature returning the event, and ***yyy*** is the message type. For example, if a TBCT operation is successful, buffer points to a structure named NCC_ISDN_TBCT_RETURN_RESULT.

Sending and receiving transparent messages

The transparent message sending and receiving feature is supported by the 4ESS, DMS, ETSI, QSIG, and NI2 variants.

This feature allows an application to include raw Q.931 data in one or more custom-built information elements (IEs) in messages sent to the stack. These information elements, called transparent IEs, are inserted in the Q.931 message generated by the stack exactly as specified by the user.

An application can also read the raw data in an incoming Q.931 message.

Sending transparent messages

To send a transparent message, the application invokes **nccSendCallMessage**. In the function invocation, the ***message*** argument must point to an NCC_ISDN_TRANSPARENT_BUFFER_INVOKE structure specifying the message. The ***size*** argument must be set to the size of this structure. The structure is:

```
typedef struct
{
    NCC_ISDN_SEND_CALL_MESSAGE hdr;
    unsigned char   isdn_message;    /* ISDN message                */
    unsigned char   size;            /* Size of the buffer          */
    char            buffer[1];       /* Buffer, attached to the msg  */
} NCC_ISDN_TRANSPARENT_BUFFER_INVOKE;
```

The structure's header contains:

- hdr.message_id: NCC_ISDN_TRANSPARENT_BUFFER
- hdr.message_type: 0

The isdn_message field in the structure contains the ISDN message:

isdn_message value	Description
NCC_ISDN_FACILITY	ISDN FACILITY message.
NCC_ISDN_NOTIFY	ISDN NOTIFY message.
NCC_ISDN_INFO	ISDN INFORMATION message.

The buffer field is a buffer containing the raw transparent message data supplied by the application in hexadecimal format. The size field contains the size of the buffer.

The following code sample shows how to create and send a raw FACILITY message using the transparent message send/receive feature:

```
void CreateRawFacility(NCC_CALLHD callhd)
{
NCC_ISDN_TRANSPARENT_BUFFER_INVOKE * p_call_message;
unsigned size;                      /* Size of message                  */

unsigned char buffer_size;          /* Size of a transparent buffer     */
char buffer[]                       /* Example of transparent buffer    */
{   0x1c , 0x06 , 0x91 , 0xa2 , 0x03 , 0x02 , 0x01 , 0x03};

buffer_size = sizeof( buffer );
                                    /* Calculate the size of the message */
size = sizeof( NCC_ISDN_TRANSPARENT_BUFFER_INVOKE ) + buffer_size;
p_call_message = (NCC_ISDN_TRANSPARENT_BUFFER_INVOKE *) malloc ( size );

memset (p_call_message, 0, size);   /* First zero out the entire buffer  */

p_call_message->hdr.message_id = NCC_ISDN_TRANSPARENT_BUFFER;
p_call_message->isdn_message = NCC_ISDN_FACILITY;
p_call_message->size = buffer_size;
memcpy(p_call_message->buffer, buffer, p_call_message->size);

if ( nccSendCallMessage( callhd, p_call_message, size ) != SUCCESS )
    printf( nccSendCallMessage failed \n );

free( p_call_message );
}
```

To use transparent IEs, the application must disable the stack's syntax checking mechanism. To do this, set the NS_IE_RELAY_BEHAVIOUR bit in the ns_behaviour sub-structure referenced in the ISDN_PROTOCOL_PARMS_CHANNELIZED structure passed to **isdnStartProtocol**. By default, this bit is 0 (zero).

The ISDN daemon *isdncta* has a command-line switch (-N) that sets the NS_IE_RELAY_BEHAVIOUR bit.

Receiving transparent messages

To receive FACILITY messages that are not recognized by the stack, the application must enable the ACU_SEND_UNKNOWN_FACILITY bit in the acu_behaviour sub-structure contained in the ISDN_PROTOCOL_PARMS_CHANNELIZED structure. By default, this bit is 0 (zero).

By setting this behavior bit, the bit NS_ACCEPT_UNKNOWN_FAC_IE is implicitly enabled. The ISDN daemon (*isdncta)* has a command-line switch (-A) that sets the ACU_SEND_UNKNOWN_FACILITY bit.

When the stack receives an unknown FACILITY message, an NCCEVN_PROTOCOL_EVENT is generated and sent to the application. The buffer field (returned with the event ID in the CTA_EVENT structure) points to an NCC_ISDN_TRANSPARENT_BUFFER_RETURN_RESULT structure. This structure has its own buffer containing the FACILITY message. The structure is:

```
typedef struct
{
    NCC_ISDN_SEND_CALL_MESSAGE hdr;
    unsigned char   isdn_message;    /* ISDN message                */
    unsigned char   size;            /* Size of the buffer          */
    char            buffer[1];       /* Buffer, attached to the msg  */
} NCC_ISDN_TRANSPARENT_BUFFER_RETURN_RESULT;
```

The structure's header contains:

- hdr.message_id: NCC_ISDN_TRANSPARENT_BUFFER
- hdr.message_type: NCC_ISDN_RETURN_RESULT

The isdn_message field is set to NCC_ISDN_FACILITY.

The size field contains the size of the Q931 message.

The buffer field contains the Q931 message.

The application is responsible for freeing the event buffer returned in the NCC_ISDN_TRANSPARENT_BUFFER_RETURN_RESULT structure. To do so, the application can invoke **ctaFreeBuffer.**

The following code sample shows how to receive and process transparent messages:

```
CTA_EVENT event = {0};
for (;;)
{
  ctaWaitEvent( ctaqueuehd, &event, 100);
  ProcessEvent(&event)
  /* If the CTA_INTERNAL_BUFFER bit is set in the size field, the
  application is responsible for freeing the memory that it did not
  allocate. */
  if ( event.size & CTA_INTERNAL_BUFFER )
    ctaFreeBuffer( event.buffer );     /* Release Natural Access buffers */
}

void ProcessEvent(CTA_EVENT *event)
{
  switch( event->id )
  {
  case NCCEVN_PROTOCOL_EVENT:
    switch ( event->value )
    {
    case ISDN_CALL_MESSAGE_SENT:
      {
      NCC_ISDN_SEND_CALL_MESSAGE *hdr;
      if ( event->size == 0 )
      {
       printf( "Failed to receive buffer with ISDN_CALL_MESSAGE_SENT\n");
       break;
      }
      if ( (hdr->message_id == NCC_ISDN_TRANSPARENT_BUFFER) &&
        (hdr->message_type == NCC_ISDN_RETURN_RESULT) )
        {
        NCC_ISDN_TRANSPARENT_BUFFER_RETURN_RESULT *retres;
        retres = (NCC_ISDN_TRANSPARENT_BUFFER_RETURN_RESULT *)event->buffer;
        if ( retres->isdn_message == NCC_ISDN_FACILITY )
           printf( Received Facility message with unknown Facility IE\n );
        }
      }
      break;                       /* End of case ISDN_CALL_MESSAGE_SENT */
   default:
      break;
   }                               /* End of switch ( event->value )     */
  break;                           /* End of case NCCEVN_PROTOCOL_EVENT  */
...
default:
  break;
  }                                /* End of switch( event->id )         */
}
```

7. Extended parameters

Using extended parameters

An application can send various types of ISDN-specific information during certain call control operations. This information includes address information, user-to-user information (UUI), and other values.

When using the NCC API, information is sent through extended parameter structures. An application can provide the extended parameter structures during the following operations:

- Call establishment (with **nccPlaceCall**)
- Call answering (with **nccAnswerCall**)
- Call accepting (with **nccAcceptCall**)
- Call rejection (with **nccRejectCall**)
- Call disconnect (with **nccDisconnectCall**)
- Call establishment (with **nccSendDigits**)
- Call transfer (with **nccTransferCall**)

In each case, the information is sent when the function call is made.

To specify the information to be sent, fill a data structure and specify p_data as the name for the pointer to the structure. Then, specify the p_data structure as the ***void ** argument in the appropriate function invocation.

Not all fields in these data structures can be used at all times. The fields valid for a given message differ depending upon the network protocol variant used and the actual ISDN message sent on the trunk in response to the function call.

Refer to the *nms\include\isdnval.h* file that contains the values for many of the fields found in extended parameter structures and the NCC_ISDN_EXT_CALL_STATUS structure.

For each of the extended parameter descriptions, tables list valid field values for each network protocol variant. The fields valid for a function depend not on the function itself, but on the ISDN message that the function sends on the trunk (for example, SETUP, CALL PROCEEDING, and so on). For **nccAnswerCall**, **nccAcceptCall**, and **nccRejectCall**, the message sent by each function depends upon the setting of the NCC.X.ADI_ISDN.START_EXT.flags parameter. Refer to ISDN TCP parameters overview for more information.

Only one message of a given type (such as CALL PROCEEDING, PROGRESS, and ALERTING) can be sent to the network. For example, if your application sets the NCC.X.ADI_ISDN.START_EXT.flags parameter such that a PROGRESS message is to be sent with **nccAcceptCall** and **nccRejectCall**, and then calls both functions, only one PROGRESS message is sent.

nccPlaceCall

This topic describes:

- PLACECALL_EXT field validity for network variants
- PLACECALL_EXT field descriptions

This is the ISDN Software **nccPlaceCall** extended parameter structure and its substructures:

```
typedef struct
{
    DWORD size;                          /* Size of this structure            */
    BYTE ie_list[NCC_ISDN_MAX_IE_LIST]; /* additional information elements   */
    CALLEDNUM      callednumber;         /* Called number substructure        */
    CALLINGNUM     callingnumber;        /* Calling number substructure       */
    REDIRECTINGNUM redirectingnumber;    /* Redirecting number substructure   */
    char callingname[32];                /* Calling name                      */
    WORD   service;                      /* Service                           */
    WORD   nsf_present;                  /* NSF usage flag                    */
    WORD   nsf_service_feature;          /* Service or feature is set in the  */
                                         /* Coding field                      */
    WORD   nsf_facility_coding;          /* NSF coding                        */
    WORD   nsf_param_fld;         /* NSF parameterized facility coding value.*/
    WORD   getcallid;                    /* Get callid for this call          */
}   PLACECALL_EXT;

typedef struct
{
    DWORD   size;                /* Size of this structure                  */
    WORD    plan;                /* Q.931 numbering plan of calling address */
    WORD    type;                /* Q.931 numbering type of calling address */
    WORD    screen;              /* Q.931 ANI screening indicator           */
    WORD    presentation;        /* Q.931 caller ID presentation indicator  */
} CALLINGNUM;

typedef struct
{
    DWORD   size;                /* Size of called number                   */
    WORD    plan;                /* Q.931 numbering plan of called address  */
    WORD    type;                /* Q.931 numbering type of called address  */
} CALLEDNUM;

typedef struct
{
    DWORD size;           /* Size of this structure                         */
    char  digits[33];     /* Redirecting address                            */
    char  pad[3];         /* Padding                                        */
    WORD  plan;           /* Q.931 numbering plan of redirecting address    */
    WORD type;            /* Q.931 numbering type of redirecting address    */
    WORD screen;          /* Q.931 redirecting address screening indicator  */
    WORD presentation;    /* Q.931 redirecting address presentation indicator*/
    WORD reason;          /* Q.931 reason for redirection                   */
    WORD pad[1]           /* Padding                                        */
} REDIRECTING_NUM;
```

PLACECALL_EXT field validity for network variants

nccPlaceCall always causes a SETUP message to be sent on the trunk. The following table shows the fields in PLACECALL_EXT that are valid for each network variant:

PLACECALL_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
ie_list	x	x	x	x	x	x	x	x	x	x	x	x	x	x
callednumber	x	x	x	x	x	x	x	x	x	x	x	x	x	x
callingname		x	x	x								x		
callingnumber	x	x	x	x	x	x	x	x	x	x	x	x	x	x
getcallid			x	x										
redirectingnumber	x	x	x	x										
service	x	x	x	x	x	x	x	x	x	x	x	x		x
nsf_param_fld	x													
nsf_present	x													
nsf_service_feature	x													
nsf_facility_coding	x													

PLACECALL_EXT field descriptions

The following table describes the fields in the PLACECALL_EXT structure and in substructures referenced by this structure. Allowed values for these fields are defined in *isdnval.h*.

Field	Description
size	Size of PLACECALL_EXT.
ie_list	Null-terminated array of additional Q.931 information elements. See Sending additional information elements for more information.
callednumber.size	Size of called number.

Field	Description
callednumber.plan	Q.931 numbering plan of called address.
callednumber.type	Q.931 numbering type of called address.
callingname	Pointer to calling name information.
callingnumber.size	Size of calling number.
callingnumber.plan	Q.931 numbering plan of calling address.
callingnumber.type	Q.931 numbering type of calling address.
callingnumber.screen	Q.931 ANI screening indicator.
callingnumber.presentation	Q.931 caller ID presentation indicator.
redirectingnumber.size	Size of redirecting address.
redirectingnumber.digits	Redirecting address.
redirectingnumber.plan	Q.931 numbering plan of redirecting address.
redirectingnumber.type	Q.931 numbering type of redirecting address.
redirectingnumber.screen	Q.931 redirecting address screening indicator.
redirectingnumber.presentation	Q.931 redirecting address presentation indicator.
redirectingnumber.reason	Q.931 reason for redirection.
service	
nsf_facility_coding	NSF service or feature ID.
nsf_param_fld	NSF parameterized facility coding value.
nsf_present	Network-specific facilities usage flag.
nsf_service_feature	Service or feature is set in the coding field.

nccAnswerCall

The **nccAnswerCall** extended parameter structure for ISDN Software is shown:

```
typedef struct
{
    DWORD size;                         /* Size of this structure          */
    BYTE ie_list[NCC_ISDN_MAX_IE_LIST]; /* additional information elements  */

} ANSWERCALL_EXT;
```

ANSWERCALL_EXT field validity for network variants

nccAnswerCall generates an ALERTING message by default. The following table shows the fields in ANSWERCALL_EXT that are valid for each network variant:

ANSWERCALL_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T160607
ie_list	x	x	x	x	x	x	x	x	x	x	x	x	x	x

ANSWERCALL_EXT field descriptions

By changing the setting of the NCC.X.ADI_ISDN.START_EXT.flags parameter, **nccAnswerCall** sends a CALL PROCEEDING or PROGRESS message instead of the ALERTING message. No uui may be sent in this case.

The following table describes the field in the ANSWERCALL_EXT structure:

Field	Description
ie_list	Null-terminated array of additional Q.931 information elements. See Sending additional information elements for more information.

nccAcceptCall

The **nccAcceptCall** extended parameter structure for ISDN Software is shown:

```
typedef struct
{
    DWORD size;                         /* Size of this structure          */
    BYTE ie_list[NCC_ISDN_MAX_IE_LIST]; /* additional information elements  */
    WORD  cause;                        /* Cause value                     */
    WORD  progressdescription;          /* Progress description            */

} ACCEPTCALL_EXT;
```

ACCEPTCALL_EXT field validity for network variants - PROGRESS message

nccAcceptCall generates a PROGRESS message by default. The following table shows the fields in ACCEPTCALL_EXT that are valid for each network variant:

ACCEPTCALL_EXT field (PROGRESS message)	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
ie_list (Not sent)									x					
cause	x	x	x	x	x									x
progressdescription	x	x	x	x	x				x					x

ACCEPTCALL_EXT field validity for network variants - ALERTING message

By changing the setting of the NCC.X.ADI_ISDN.START_EXT.flags parameter, **nccAcceptCall** sends a CALL PROCEEDING or ALERTING message instead of the PROGRESS message. In the case of CALL PROCEEDING, no extended parameters can be sent. In the case of an ALERTING message, the UUI and progressdescription fields are valid for certain variants:

ACCEPTCALL_EXT field (ALERTING message)	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
ie_list	x	x	x	x	x	x	x	x	x	x	x	x	x	x
cause (Not sent)														
progressdescription		x	x	x	x	x			x	x	x			x

ACCEPTCALL_EXT field descriptions

The following table describes the fields in the ACCEPTCALL_EXT structure:

Field	Description
size	Size of ACCEPTCALL_EXT.
ie_list	Null-terminated array of additional Q.931 information elements. See Sending additional information elements for more information.
cause	Cause field for the associated PROGRESS message.

Field	Description
progressdescription	PROGRESS description.

nccRejectCall

The **nccRejectCall** extended parameter structure for ISDN Software is shown:

```
typedef struct
{
    DWORD size;                         /* Size of this structure           */
    BYTE ie_list[NCC_ISDN_MAX_IE_LIST]; /* additional information elements  */
    WORD  cause;                        /* Disconnect cause (NCC value)     */
    WORD  pad;

} REJECTCALL_EXT;
```

REJECTCALL_EXT field validity for network variants

The following table shows the fields in REJECTCALL_EXT that are valid for each network variant:

REJECTCALL_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
ie_list	x	x	x	x	x	x	x	x	x	x	x	x	x	x
cause	x	x	x	x	x	x	x	x	x	x	x	x	x	x

nccRejectCall generates a PROGRESS message by default. If this message is sent, only the cause field is used.

By changing the setting of the NCC.X.ADI_ISDN.START_EXT.flags parameter, **nccRejectCall** sends a CALL PROCEEDING or ALERTING message instead of the PROGRESS message. With ALERTING, only the uui field is sent. With CALL PROCEEDING, no extended parameters can be sent.

REJECTCALL_EXT field descriptions

The following table describes the fields in the REJECTCALL_EXT structure:

Field	Description
size	Size of REJECTCALL_EXT.
ie_list	Null-terminated array of additional Q.931 information elements. See Sending additional information elements for more information.
cause	Cause for call rejection.

nccDisconnectCall

The **nccDisconnectCall** extended parameter structure for ISDN Software is shown:

```
typedef struct
{
    DWORD   size;                          /* Size of this structure          */
    BYTE ie_list[NCC_ISDN_MAX_IE_LIST];    /* additional information elements */
    WORD    cause;                         /* Disconnect cause (NCC value)    */
    WORD    pad;

} DISCONNECTCALL_EXT;
```

DISCONNECTCALL_EXT field validity for network variants

nccDisconnectCall generates a DISCONNECT message by default. The following table shows the fields in DISCONNECTCALL_EXT that are valid for each network variant:

DISCONNECTCALL_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
ie_list	x	x	x	x	x	x	x	x	x	x	x	x	x	x
cause	x	x	x	x	x	x	x	x	x	x	x	x	x	x

DISCONNECTCALL_EXT field descriptions

The following table describes the fields in the DISCONNECTCALL_EXT structure:

Field	Description
size	Size of DISCONNECTCALL_EXT.
ie_list	Null-terminated array of additional Q.931 information elements. See Sending additional information elements for more information.
cause	Cause for disconnect.

nccSendDigits

The **nccSendDigits** extended parameter structure for ISDN Software and its substructure is shown:

```
typedef struct
{
    DWORD     size;              /* Size of this structure               */
    CALLEDNUM callednumber;      /* Called number substructure           */
} SENDDIGITS_EXT;

typedef struct
{
    DWORD   size;                /* Size of called number                 */
    WORD    plan;                /* Q.931 numbering plan of called address */
    WORD    type;                /* Q.931 numbering type of called address */

} CALLEDNUM;
```

SENDDIGITS_EXT field validity for network variants

nccSendDigits generates an INFORMATION message by default. The following table shows the fields in SENDDIGITS_EXT that are valid in this case for each network variant:

SENDDIGITS_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T16007
callednumber				x	x	x				x	x	x		

SENDDIGITS_EXT field descriptions

The following table describes the fields in the SENDDIGITS_EXT structure:

Field	Description
callednumber.size	Size of the SENDDIGITS_EXT structure.
callednumber.plan	Q.931 numbering plan of called address.
callednumber.type	Q.931 number type of called address.

nccTransferCall

The **nccTransferCall** extended parameter structure for ISDN Software is shown:

```
typedef struct
{
    DWORD size;                        /* Size of this structure              */
    WORD id[NCC_ISDN_CALLID_LEN];      /* Call identifier                     */

} TRANSFERCALL_EXT;
```

TRANSFERCALL_EXT field validity for network variants

The following table shows the field in TRANSFERCALL_EXT that is valid in this case for each network variant:

TRANSFERCALL_EXT field	4ESS	E10	NI2	DMS	ETSI	VN6	HKG	AUSTEL1	NTT	KOREA	TAIWAN	QSIG	DPNSS	T1607
id			x	x	x							x	x	

TRANSFERCALL_EXT field descriptions

The following table describes the fields in the TRANSFERCALL_EXT structure:

Field	Description
size	Size of the TRANSFERCALL_EXT structure.
id	Call identifier of the second call.

Sending additional information elements

Applications can add Q.931 information elements, including user-to-user information, to some ISDN messages.

The array, ie_list in the extended parameter structure, can be filled with fully formed Q.931 information elements to be passed transparently through the stack. This list must be terminated with a null (0x00) byte.

For example, to send user-to-user information elements using the ie_list array:

```
ie_list[0] = 0x7E;  /* user-to-user ie codepoint                  */
ie_list[1] = 0x03;  /* length of user-to-user ie                  */
ie_list[2] = 0x04;  /* user-to-user data = IA5                    */
ie_list[3] = 'A'
ie_list[4] = 'B'
ie_list[5] = 0x00;  /* end of additional information elements     */
                    /* note that this is not the string terminator */
                    /* for the user-to-user IA5 data.             */
```

Receiving user-to-user information

User-to-user information (UUI) can be presented to the application at any time during a call. When user-to-user information is received, an NCCEVN_EXT_CALL_STATUS_UPDATE event is generated. The value field contains CALL_STATUS_UUI.

The UUI information is available in the NCC_ISDN_EXT_CALL_STATUS structure. It can be retrieved using **nccGetExtendedCallStatus**.

The application receives the entire information element, including the:

- UUI IE identifier
- UUI IE length
- UUI IE protocol discriminator
- User-defined data

Receiving charging information

Charging information is contained within the NCC_ISDN_EXT_CALL_STATUS structure, and can be accessed by invoking **nccGetExtendedCallStatus**. Charging information can be sent to an application after a call has reached the connected state.

Charging is supported only for the TE side of the VN6/FRANCE variant.

8. ISDN B channel assignment

B channel assignment overview

When a TCP instance is launched, a B channel is assigned to it. This channel is called the default channel. When placing a call, the TCP requests the use of the default channel from the network. It directs the network to respond in one of two ways, or modes:

Mode	Description
Exclusive	If the TCP's default B channel is not available on the network, the network clears the call.
Non-exclusive (also called preferred mode)	If a TCP's default B channel is not available on the network, the network directs the TCP to use another channel. In this case, the application must perform switching to connect the new B channel with the DSP resource associated with the TCP.

In the US, ISDN systems almost always place and receive calls in exclusive mode. In Europe, both modes are used: many central office (CO) switches require the TE equipment side to use non-exclusive mode.

For details about the switch model for a board, see the installation and developer's manual for the board.

Default channel assignment

A default channel is assigned to each TCP instance in either of two ways.

Ordinarily, the network stream and timeslot are inferred from the DSP stream and timeslot that were specified when the context was opened. This is the trunk slot that is connected to the DSP slot when the Clocking.HBus.ClockMode board keyword is set to STANDALONE. See Making switch connections for ISDN Software for more information.

The default channel for the NCC API can be specified by setting the following board keywords for that TCP instance. These parameters are documented in ISDN Software parameter files.

- NCC.ADI_ISDN.START_EXT.networkstream
- NCC.ADI_ISDN.START_EXT.networkslot

Exclusive mode

In exclusive mode, a TCP instance effectively owns its default channel. To place an outbound call, the TCP asks the network for the default channel. If the channel is not available, the network clears the call. When the TCP senses this, NaturalAccess sends NCCEVN_CALL_DISCONNECTED to the application.

For example, the application attempts to establish an outbound call using a TCP that is set up to use default B channel 5 in exclusive mode (corresponding to MVIP-95 0:4, 1:4). The TCP asks the network for B channel 5. If the channel is available, the network signals the TCP to continue with the call on that channel. Otherwise, the network clears the call. In MVIP-90 terms, this corresponds to 16:0.

In exclusive mode, any incoming call will be routed to a TCP instance only if the call is on its default channel. All calls on that channel are routed to the TCP.

To set a TCP instance to exclusive mode, set the NCC.ADI_ISDN.START_EXT.exclusive parameter to 1.

You can also use the NCC.ADI_ISDN.START_EXT.direction parameter to specify whether the channel is incoming-only, outgoing-only, or bidirectional. Incoming calls are never offered to an outgoing-only TCP channel. A call placed on an incoming-only channel is rejected.

The direction parameter setting must match the configuration of the connected equipment (for example, the network switch). If the application attempts to place a call on a channel that is configured on the connected equipment to be incoming-only, the outbound call is always rejected.

Note: DPNSS only operates in exclusive mode.

Non-exclusive mode (preferred mode)

When a TCP instance places a call in non-exclusive mode, it requests the use of its default channel from the network. The network allows the use of the default channel or directs the TCP to use a different channel. If a different channel is assigned, the application must perform switching to route the call to the DSP resource associated with the TCP instance.

To do so, the application must:

- Determine which B channel is chosen by the network for the TCP (for example, the default channel or another channel). If no channel is available, the call is rejected by the network.

  When the application receives the NCCEVN_PLACING_CALL event, the stream and timeslot corresponding to the channel are available in the NCC_ISDN_EXT_CALL_STATUS structure. To access the stream and timeslot, invoke **nccGetExtendedCallStatus.**

- Connect the stream and timeslot for that channel with the stream and timeslot assigned to the context on which the call is placed.

  To perform this connection, the application can use the NaturalAccess Switching API. The switching should be performed as soon as possible after NCCEVN_PLACING_CALL is received, to ensure that call progress analysis succeeds. For an example of applications that perform this switching, see the *isdnncc* sample program.

For example, if a TCP attempts to establish an outbound call on its default B channel, the default channel is 5, corresponding to MVIP-95 0:4,1:4. The network indicates that channel 5 is not available, but channel 9 is available (corresponding to MVIP-95 0:8,1:8). Since the TCP is running in non-exclusive mode, the call is established on channel 9. The application must now connect the TCP instance's DSP resource (MVIP-95 16:4, 17:4) with the B channel to establish the voice path (for example: 0:8 => 17:4, 16:4 => 1:8). In MVIP-90 terms, this connection is 16:8 <=> 18:4.

Incoming calls from any B channel are routed to the TCP instance unless the B channel on which the call is located is assigned to another TCP running in exclusive mode. For details, see Switching considerations for disconnect handling.

To set a TCP instance to exclusive mode, set the NCC.ADI_ISDN.START_EXT.exclusive parameter to 1.

Switching considerations for disconnect handling

In non-exclusive mode, when a call is disconnected the application must disconnect only the trunk-to-DSP side of the full duplex switching connection that it made when the call was established. The DSP-to-trunk connection must not be disconnected because it may interfere with a new call arriving on the B channel that is handled by another thread or process.

Disconnecting the trunk-to-DSP connection ensures that if a new outgoing call is placed on the context, any call progress analysis functions receive only silence until the application makes a new connection.

Assigning incoming calls to TCP instances

When a call arrives on a B channel, a TCP instance is assigned to the channel as follows:

1. If a TCP instance is currently assigned to the channel in exclusive mode (and is not set up as outgoing-only), the call is routed to the TCP.

   If the TCP instance is outgoing-only, the call is rejected.
2. If a TCP is currently assigned to the channel in non-exclusive mode (for example, the channel is the TCP's default channel), the call is routed to the TCP if it is not already handling another call (and is not set up as outgoing-only).
3. Otherwise, the call is passed to any free TCP that is in non-exclusive mode and is not set up as outgoing-only.
4. If no TCP is available, the call is rejected.

9. Demonstration programs

Demonstration programs overview

Each demonstration program is shipped as an executable program, including its source and makefiles. Before you start the demonstration programs, ensure that:

- NaturalAccess is properly installed.
- The board is executing.
- Switching is correctly configured.

The following demonstration programs are provided with ISDN Software:

Program	Description
isdncta	A daemon that starts and stops the ISDN protocol stack.
isdnncc	ISDN NaturalCallControl demonstration program.

isdncta (ISDN daemon)

Starts and stops the ISDN protocol stack. Demonstrates:

- Using NaturalAccess to start and stop the ISDN protocol stack.
- Performing switching connections necessary to support ISDN Software.

Once the ISDN protocol stack is started using this program, you can run any other NaturalAccess call control program with ISDN Software.

Featured functions

isdnStartProtocol, **isdnStopProtocol**

Requirements

- A digital trunk interface board.
- NaturalAccess installed.

Usage

```
isdncta [options]
```

where ***options*** is one or more of the following:

Option	Description	Default
-a ***nai***	Network access identifier of trunk to use.	0
-g ***group_#***	NFAS group number if multiple CCID is configured.	
-b ***board_#***	Board number (terminal equipment interface). Do not use -B if this option is specified.	0

Option	Description	Default
-B ***board_#***	Board number (network terminator interface). Do not use -b if this option is specified.	0
-I ***hex_bit***	(Optional) Setting for in_calls_behaviour, in hexadecimal. Use to make bit settings required for overlap receiving and/or channel ID setting: CC_TRANSPARENT_OVERLAP_RCV = 0x0080 CC_SET_CHAN_ID = 0x2000	0x0
-o ***operator***	Network operator variant. Allowed values: 3 = France Telecom VN6 8 = Northern Telecom DMS 100 9 = INS-1500 NTT 11 = EuroISDN 15 = Australian Telecom 1 16 = QSIG 17 = Hong Kong Telephone 20 = US National ISDN 2 23 = AT&T 5ESS10 24 = AT&T 4ESS 25 = Korea 50 = Taiwan 51 = DPNSS Use EuroISDN (11) and Australian Telecom (15) for the following countries: Austria, Denmark, Finland, Greece, Iceland, Ireland, Italy, Liechtenstein, Luxembourg, Netherlands, Norway, Portugal, Russia, Spain, and Switzerland.	23 (AT&T 5ESS10)
-N ***hex_bit***	(Optional) Setting for ns_behaviour, in hexadecimal. Determines the responses of the NS layer. Use to make bit settings required for allowing sending and receiving of ISDN-specific messages: NS_ACCEPT_UNKNOWN_FAC_IE =0x0004 NS_IE_RELAY_BEHAVIOUR =0x0010 NS_SEND_USER_CONNECT_ACK =0x0100 (ETSI only) NS_EXPLICIT_INTERFACE_ID =0x0200 NS_PRESERVE_EXT_BIT_IN_CHAN_ID =0x0400 (DMS only)	0x0

Option	Description	Default
-A ***hex_bit***	(Optional) Setting for acu_behaviour, in hexadecimal. Use to make bit setting required (by ETSI, Q.SIG, and NI2 variants) to enable the stack to forward to the application ISDN-specific messages containing unrecognized facility IEs. See Sending and receiving ISDN-specific messages: ACU_SEND_UNKNOWN_FACILITY =0x0004	0x0
-O ***hex_bit***	(Optional) Setting for out_calls_behaviour, in hexadecimal. Use to make bit settings required for overlap sending and/or A-law/mu law coding. See Sending and receiving ISDN-specific messages: CC_USER_SENDING_COMPLETE =0x0002 CC_USE_MU_LAW=0x0010 CC_USE_A_LAW=0x0020 CC_E1_CONTINUOUS_CHANNELS=0x40 CC_E1_CONTINUOUS_CHANNELS_LOGICAL=0x400 CC_SET_CALL_ID_TO_CRV=0x80 CC_USE_PATH_REPLACEMENT=0x100 CC_USE_SINGLE_STEP_TRANSFER=0x800	0x0
-c ***country***	Country. The behavior of a network operator can change depending on the country specified. The default country depends on the operator. Allowed values: 1 = USA 32 = Belgium 33 = France 44 = Great Britain 46 = Sweden 49 = Germany 61 = Australia 81 = Japan 82 = Korea 86 = China 92 = Singapore 52 = Hong Kong 866 = Taiwan 1000 = Europe **Note:** Use Europe (1000) for the following countries: Austria, Denmark, Finland, Greece, Iceland, Ireland, Italy, Liechtenstein, Luxembourg, Netherlands, Norway, Portugal, Russia, Spain, and Switzerland.	1 (USA)

Option	Description	Default
-s	Creates the default connections to support ISDN call control. Only necessary if switching is enabled in the board keyword file (such as, Clocking.HBus.ClockMode = MASTER_A, MASTER_B, or SLAVE).	No connections are made. (Assumes switching is disabled.)
-T	t309 option for D channel backup configurations. See the *Dialogic® NaturalAccess™ ISDN Software Installation Manual* for information on D channel backup.	Disabled.
-h or -?	Display a help screen and terminate the program.	N/A

Functional overview

To enable other NaturalAccess programs to place or receive calls on an ISDN trunk, start the ISDN protocol stack by running *isdncta*. This program allows you to choose from several ISDN operator variants based on command line options.

When *isdncta* is launched, it performs the following actions:

Step	Action
1	Parses command line arguments and ensures that all arguments are valid and do not conflict.
2	Initializes NaturalAccess.
3	If the -s option is specified, *isdncta* makes connections needed to support ISDN call control. (If Clocking.HBus.ClockMode = MASTER_A, MASTER_B, or SLAVE in the configuration file, these connections are made automatically when the board boots.) For more information, see Making switch connections for ISDN Software.
4	If the -g option is specified, *isdncta* sets ENABLE_MULTIPLE_CCID parameters in the ISDN_PROTOCOL_PARMS_CHANNELIZED structure to enable Multiple CCID configuration, and passes the provided NFAS group number as an extra argument to **isdnStartProtocol**.
5	Starts the ISDN protocol stack on the board specified on the command line, either as Terminal Equipment (TE) or as Network Equipment (NT), with the specified network operator variant and country variant.
6	Waits for **Q** to be pressed. When the **Q** key is pressed, the daemon program stops the ISDN protocol stack and terminates.

Procedure

Complete the following steps to launch *isdncta*:

Step	Action
1	Set up the configuration file to describe the board and software. For more information, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual.*
2	Run *oamsys* to initialize the hardware and to make the configuration file changes effective.
3	To start the ISDN protocol stack, enter: `isdncta [options]` where ***options*** is one or more of the command line options as previously described.
4	Launch another demonstration program.
5	When you have finished, press **Q** to terminate *isdncta*.

Compilation

isdncta is supplied in executable form and as source code. If you need to recompile *isdncta*, do one of the following:

Under this OS...	Go to this directory...	Enter...
Windows	*\nms\ctaccess\demos\isdncta*	nmake
UNIX	*/opt/nms/ctaccess/demos/isdncta*	make

For more information, see the ISDN Software readme file.

isdncta structure and coding features

isdncta is based on the NaturalAccess asynchronous programming model. The NaturalAccess function names start with ncc, adi, or cta. For example, **nccStartProtocol** starts a TCP instance on a context. Other functions in *isdncta* are found in CTADEMO, a library distributed with NaturalAccess containing examples of how to use the API. CTADEMO functions have names starting with Demo. One example is **DemoOpenPort**, which opens the NaturalAccess application queue, and then calls another demonstration function to create a context and open services on the new queue. Most CTADEMO functions are wrappers that enclose the corresponding NCC function and wait for an event to terminate the function, signaling whether the function was successful. In this way, they transform an asynchronous wait for an event into a synchronous function.

main function

The **main** function of *isdncta* defines the list of NaturalAccess service managers that the application needs. It calls **ParseArguments**, which parses command line arguments, and assigns the corresponding values to variables. **ParseArguments** also checks that none of the user options are inconsistent with each other, and sends a warning if it finds a problem.

main registers an error handler with NaturalAccess, calls **ctaInitialize** to initialize NaturalAccess, and calls **MyOpenPort**, which uses the demonstration library function **DemoOpenPort** to open a context.

If the -s option was specified, **main** calls **MyMakeConnection**, which uses the NaturalAccess Switching API to make the connections necessary to support Natural Call Control. For more information, see Making switch connections for ISDN Software.

main calls **IsdnStart** to start the ISDN protocol stack. **IsdnStart** calls **isdnStartProtocol** on the open context with the operator and country variant specified on the command line. **isdnStartProtocol** is started with ISDN_PROTOCOL_CHANNELIZED specified, so it runs in Natural Call Control mode.

main calls **WaitForKeyboardEvent**. **WaitForKeyboardEvent** waits for the **Q** key to be pressed. When the key is pressed, **main** calls **IsdnStop**, which calls **isdnStopProtocol** to stop the ISDN protocol stack. Then the daemon program terminates.

isdnncc (ISDN NCC call control demonstration)

Uses the ISDN TCP with NaturalAccess to receive and place calls.

Demonstrates:

- Using NCC functions
- Operating the ISDN TCP on a live trunk
- Operating of the ISDN TCP across the CT bus
- Operating in non-exclusive mode

isdnncc requires prior initialization of the ISDN protocol stack with **isdnStartProtocol**. To initialize the ISDN protocol stack, a program similar to *isdncta* can be used.

Featured functions

adiCollectDigits, adiGetBoardInfo, adiStartCallProgress, adiStartDTMF, **adiStartTones**, **nccAcceptCall**, **nccAnswerCall**, **nccDisconnectCall, nccGetCallStatus**, **nccGetExtendedCallStatus, nccPlaceCall**, **nccQueryCapability, nccRejectCall**, **nccReleaseCall, nccStartProtocol, swiMakeConnection**, **vcePlayList**, **vceRecordMessage**

Requirements

- A digital trunk interface board.
- NaturalAccess installed.
- *isd0.tcp* TCP file.
- Country-specific parameter files (*.pf* files).
- *nccxisdn.par* (optional).

Usage

isdnncc [***options***]

where ***options*** is one or more of the following:

Option	Description	Default
-a ***mode***	Accepts a call, without answering. ***mode*** specifies the call acceptance method: 1 - Accepts and plays ring back tone. 2 - Accepts and remains silent until next command. 3 - Accepts with user audio and plays a voice file. 4 - Accepts with user audio and detects DTMFs.	Answers the call.
-A ***string***	Specifies data for the ie_list field in **nccAcceptCall**.	Disabled
-b ***board***_#	Specifies a board number.	0
-C ***string***	Specifies data for the ie_list field in **nccAnswerCall**.	Disabled
-d	Uses the TCP's default parameters, without loading the parameter file. Refer to ISDN TCP parameters overview for details on the default NCC parameters of the ISDN Software TCP.	Loads the parameter file.
-D ***number***	Sets the calling party number for outbound calls.	987
-F	Sends the calling party name	Disabled
-h or -?	Displays a Help screen and terminates.	N/A
-i	Awaits incoming calls.	Awaits outbound calls.
-I ***string***	Specifies data for the ie_list field in an INFORMATION message to be sent after callconnect.	Disabled
-n ***num_to_dial***	Sets the number to dial. ***num_to_dial*** must be formatted appropriately.	123

Option	Description	Default
-N ***string***	Specifies data for the ie_list field in a NOTIFY message to be sent after callconnect.	Disabled
-p	Determines by the first digit of the called number if the inbound call should be rejected.	Disabled
-P	Allows the user to place a call on an ISDN PRI trunk in Non-exclusive mode. This mode allows the equipment on the other side of the trunk to accept a call on any B channel. Some ISDN network terminators do not accept exclusive calls. In these cases, you must use the -P option.	Exclusive mode (preferred mode)
-r ***rings***	Specifies the number of ring tones to play before answering an inbound call.	2
-R	Sends a redirecting number in an outbound call setup.	Disabled
-s ***n***:***m***	Specifies the stream and timeslot.	MVIP-95: 0:0 MVIP-90: 18:0 (depends upon board type)
-S ***string***	Specifies data for the ie_list field in **nccPlaceCall**.	Disabled
-t ***threads***	Specifies the number of threads to launch.	0
-u	Sends a string of user-to-user information.	Disabled
-v ***level***	Specifies the verbosity level of messages printed on screen. ***level*** can be any of the following: 0 - Displays error messages only. 1 - Displays errors and unexpected high-level events. 2 - Displays errors and all high-level events. 3 - Displays errors and all events.	2

Procedure

Complete the following steps to start *isdnncc* for an ISDN demonstration:

Step	Action
1	Start *isdncta*, as described in isdncta (ISDN daemon).
2	Enter the following command at the command line: `isdnncc [`***`options`***`]` where ***options*** is one or more of the command line options described in the previous table.

Functional overview

When *isdnncc* is started, it performs the following actions:

Step	Action
1	If the -d option is not specified, the program loads parameters that *isd0.tcp* will use. To do so, it loads and parses one of the *nccxisdn.par* parameter files provided with the product.
2	It starts the ISDN Software TCP *isd0.tcp*, configuring it using the specified parameters.
3	It either places a call or waits for an inbound call, depending on the -i command line option.
4	If it is commanded to dial out (outbound behavior), it dials automatically.

Step	Action
5	If the demonstration program is commanded to wait for a call (inbound behavior): • It waits for incoming calls. • When a call arrives, call status structures are accessed to retrieve the incoming call information such as the called number, calling number, or UUI, if any. • If the user has selected the -P command line option (non-exclusive mode), the program performs the switching to connect the B channel on which the call has been received to the appropriate DSP resource on the board. • If the -a option is specified in the command line, the call is accepted using the mode supplied by the user. • If the -p option is specified on the command line, the program checks the first digit of the incoming number. If the digit is a 7, it rejects the call and plays a special information tone (SIT). If the digit is an 8, it rejects the call and plays a busy tone. If the digit is a 9, it rejects the call, plays a reorder tone, and goes back to waiting for calls. If the digit is a 0, it immediately rejects the call by calling **nccDisconnectCall**, starting the disconnect procedure. • If a reject digit was not found in the incoming number, the program waits until a specified number of rings have been played by the TCP. When the rings have been played, the TCP answers the call. • If the first digit is not a 7, 8, 9, or 0, the demonstration program plays a welcome message and speaks back the digits it received. • It then plays another prompt asking the user to choose an action by pressing a key on the telephone keypad. Available actions are: record a file (1), play a file (2), or hang up (3). • The demonstration program starts the DTMF tone detector and waits for a tone. If the tone does not arrive, it hangs up. • If the tone arrives, the demonstration program performs the action that the tone specifies, then hangs up.

Step	Action
6	If the demonstration program seizes the line first (outbound behavior): • It dials the number specified by the user. The calling number is precoded as 987. • If the user has selected the -P command line option (non-exclusive mode), the program performs the switching to connect the B channel on which the call has been placed to the appropriate DSP resource on the board. • When the call is answered, the demonstration program starts call progress. • If call progress detects the busy tone, the program hangs up and ring tones are reported to the user. If call progress finishes with silence or detects voice, the program starts recording. • When silence is detected by the recording functions, the demonstration program stops recording and plays DTMF tone 3 (hang up) for its inbound counterpart. • When playing is completed, it hangs up.

The demonstration program hangs up if the caller hangs up at any time. If the program receives NCCEVN_CAPABILITY_UPDATE (the state of the ISDN stack has changed), it displays the new capability mask.

If the program receives NCCEVN_EXTENDED_CALL_STATUS_ UPDATE that is sent by the TCP when a new UUI string or a progress descriptor is received from the network, a new field displays on the screen.

If -v is specified on the command line, the demonstration program displays the NaturalAccess messages it receives, according to the specified verbosity level.

The following table lists the interactions between two *isdnncc* applications connected back-to-back, in which one acts as an inbound application, and the other as an outbound application:

Outbound		Inbound
Place call.	→	Wait for call.
Get connected.	←	Answer call.
Record.	←	Speak prompt and digits.
Send DTMF tone	→	Detect DTMF tone.
Hang up.	↔	Hang up.

Compilation

isdnncc is supplied as an executable as well as source code. If you need to recompile *isdnncc*, do one of the following:

Under this OS...	Go to this directory...	Enter...
Windows	*nms**ctaccess**demos**isdnncc*	nmake
UNIX	*/opt/nms/ctaccess/demos/isdnncc*	make

For more information, see the ISDN Software readme file.

isdnncc structure and coding features

isdnncc is based on an asynchronous programming model. The NaturalAccess function names start with ncc (for call control), adi (for media operations) or cta. An example is **nccGetCallStatus**, which retrieves a structure containing some information about the current call. Other function calls in *isdnncc* are found in CTADEMO, a library distributed with NaturalAccess containing examples of how to use the API. CTADEMO functions have names starting with Demo. One example is **DemoSetReportLevel**, which sets a filter to tell which messages to report. Most CTADEMO functions are wrappers that enclose the corresponding NCC, ADI, or CTA function and wait for an event to terminate the function, signaling whether the function was successful or not. In this way, they transform an asynchronous wait for an event into a synchronous function.

main function

The **main** function of *isdnncc* defines the list of NaturalAccess service managers that the application needs. It parses its command line arguments, and assigns the corresponding values to its variables. The function also checks that none of the user options are inconsistent with each other, and sends a warning if it finds a problem. Then it registers an error handler with NaturalAccess and starts NaturalAccess.

Then **main** launches the demonstration loop.

Opening the AG driver, context, and protocol

In *isdnncc*, **RunDemo** defines the list of NaturalAccess APIs needed by the application. It calls the **DemoOpenPort** function, from the CTADEMO library. **DemoOpenPort** opens the NaturalAccess application queue, attaching all defined service managers. It creates a context and opens the defined services on the context specified by the user on the specified board. If the user did not specify these parameters in the command line, the default context opened is 4:0, 5 on board 0. If this context is already in use, the call to NaturalAccess fails and the demonstration program terminates. In MVIP-90 terms, this is local streams 18:0,0.

Next, **RunDemo** calls **ctaLoadParFile**, which loads the *nccxisdn.par* parameter file containing parameters to configure the TCP. It looks for this file first in the current directory, and then in one of the following locations:

Operating system	Directory
Windows	*nms**ctaccess**cfg*\
UNIX	*/opt/nms/ctaccess/cfg/*

If the *nccxisdn.par* parameter file cannot be found, the demonstration program terminates.

RunDemo starts the TCP on the opened context. To do so, it calls **nccStartProtocol** with the following arguments:

Argument	Description
startparms	A pointer to NCC_START_PARMS.
mgrstartparms	A NULL pointer, to use the default manager parameters just loaded from the country-specific parameter file (**.pf*).
protstartparms	A pointer to NCC_ADI_ISDN_PARMS.isdn_start.

Since **RunDemo** changes the parameter defaults for NCC ***startparms*** and ISDN ***protstartparms***, it first calls **ctaGetParms** to obtain the default parameter values.

Calling MyReceiveCall or MyPlaceCall

RunDemo is ready to accept incoming calls or place outgoing calls. **MyReceiveCall** is called repeatedly. This function performs all the call interactions, and hangs up the line when the call is completed. If the -i option was not specified, the loop immediately calls **MyPlaceCall** to place a call.

Call receiving function MyReceiveCall

MyReceiveCall waits for an NCCEVN_INCOMING_CALL event and immediately queries NaturalAccess for information about the incoming call by invoking **nccGetCallStatus and nccGetExtendedCallStatus**. The attributes are:

- The called party information
- The calling party's information (ANI - automatic number identification digits)
- The redirecting party information, if present
- User-to-user (UUI) information, if present
- If non-exclusive mode was selected, the stream and timeslot to be connected with the DSP resource managed by the application

These attributes are stored in data structures, of type NCC_CALL_STATUS and NCC_ISDN_EXT_CALL_STATUS. See Retrieving call information for more information.

If non-exclusive mode was selected, **MyReceiveCall** calls **ConnectBChannel** to connect the appropriate DSP resource on the board to the B channel on the trunk.

While waiting for NCCEVN_INCOMING_CALL, **MyReceiveCall** can receive NCCEVN_RECEIVED_DIGIT, indicating that a digit has arrived in the queue. **MyReceiveCall** displays a message on the screen that indicates the digit. An NCCEVN_RECEIVED_DIGIT event is possible if the ISDN variant supports overlapped receiving mode and the user sets the appropriate behavior bit while starting ISDN protocol. See Overlapped sending and receiving for more information. **MyReceiveCall** can also receive NCCEVN_CALL_DISCONNECTED, indicating that the calling party has hung up. **MyReceiveCall** calls **hangup_in** to hang up the call.

If the state of the ISDN stack changes, **MyReceiveCall** receives NCCEVN_CAPABILITY_UPDATE. It invokes **nccQueryCapability** to retrieve the new capabilitymask. It prints this information on the screen. If the NCC_ISDN_EXT_CALL_STATUS structure is updated, NCCEVN_EXTENDED_CALL_STATUS_UPDATE is sent to the application. The value field of the event indicates the type of information that was changed. There are two fields supported by the demonstration program that can be updated: uui and progressdescription. They are changed if the value of the event is equal to the value of NCC_X_STATUS_INFO_UUI and/or the NCC_X_STATUS_INFO_PROGRESS descriptor.

MyReceiveCall decides to answer or reject a call based on the first digit in the called number. Depending upon the digit, **MyReceiveCall** answers the call or rejects it in different ways. The digits **MyReceiveCall** looks for are defined in *isdnncc.c*. They signify the following:

- If the first digit equals REJECT_BUSY_DIGIT, **MyReceiveCall** rejects the call. To do so, it calls **MyRejectCall**, which calls **nccRejectCall** with the mode NCC_REJECT_PLAY_BUSY, directing it to signal that the line is busy. **MyRejectCall** waits for the event signifying that the caller has hung up (NCCEVN_CALL_DISCONNECTED).
- If the first digit equals REJECT_REORDER_DIGIT, **MyReceiveCall** rejects the call using **nccRejectCall** with the mode NCC_REJECT_PLAY_REORDER, directing the TCP to play the reorder tone.
- If the first digit equals REJECT_SIT_DIGIT, **MyReceiveCall** rejects the call. To do so, it calls **MyRejectCall**, which calls **nccRejectCall** with the mode NCC_REJECT_USER_AUDIO. It calls **adiStartTones** to play the special information tone.
- If the first digit equals REJECT_IMMEDIATE_DIGIT, **MyReceiveCall** immediately rejects the call by calling **MyHangUp** to initiate the disconnect procedure.
- If the first digit of the called number does not correspond to any of these digits, **MyReceiveCall** answers the call. To do so, it calls **nccAnswerCall** a first time with an argument that causes the TCP to play two rings before answering automatically. **MyReceiveCall** waits for a user's keystroke.

If **MyReceiveCall** answers the call, it implements the following user interaction. This allows the demonstration program to talk either with a human being on a telephone, or with a corresponding outbound application:

- It plays a voice message welcoming the user. To play voice files, it calls **MyPlayMessage**, which opens a voice file and calls **vcePlayList** to play it.
- **MyReceiveCall** plays voice files announcing the called number and the calling number, if available.
- It plays another voice file with a menu, prompting the user to enter a DTMF tone from the telephone keypad to do one of the following: record a voice file, play a previously recorded voice file, or hang up.
- It waits for a DTMF tone to be detected. To do so, it calls **adiCollectDigits** twice in an attempt to get the digits. If a DTMF tone is not detected, **MyReceiveCall** hangs up.
- If a DTMF tone is detected, **MyReceiveCall** performs the task corresponding to the tone, and then hangs up. (If the command was to hang up, it hangs up immediately.) If the voice file to be played does not exist, **MyReceiveCall** plays a default voice file ("There is nothing to play.").

Call placement function MyPlaceCall

The function **MyPlaceCall** does the following:

- It tries to place a call to the number that the user specified on the command line.
- If -u was specified on the command line, **MyPlaceCall** fills in the PLACECALL_EXT structure using the corresponding macros and supplies the p_data pointer to the structure as an argument to **nccPlaceCall**. To place the call, **MyPlaceCall** invokes **nccPlaceCall** with the digits to call, and calling digits. It then waits for events.
- When the event NCCEVN_PLACING_CALL occurs, if non-exclusive mode was selected, the function calls **ConnectBChannel** to connect the appropriate DSP resource on the board to the B channel on the trunk.
- When the call is answered, **MyPlaceCall** starts recording the input that is on the line, waiting for a period of silence to stop recording. This operation is designed both to interact with a human being ("Hello") and to interact with **MyReceiveCall**, that speaks a little longer. To record the voice data, **MyPlaceCall** opens a voice file to receive the recording and then invokes **vceRecordMessage**. When silence is detected, **MyPlaceCall** sends a DTMF tone on the line, by invoking **adiStartDTMF**. This is strictly to interact with **MyReceiveCall**. The DTMF tone is the one that tells it to hang up. It is not disturbing for a human ear.

  MyPlaceCall can also receive NCCEVN_CALL_DISCONNECTED, indicating that the called party has hung up. **MyPlaceCall** calls **MyHangUp** function to hang up the call.

If the state of the ISDN stack changes, **MyPlaceCall** receives NCCEVN_CAPABILITY_UPDATE. It then invokes **nccQueryCapability** to retrieve the new capabilitymask. It prints this information on the screen. If the NCC_ISDN_EXT_CALL_STATUS structure is updated, NCCEVN_EXTENDED_CALL_STATUS_UPDATE is sent to the application. The value field of the event indicates the type of information that was changed. There are two fields supported by the demonstration program that can be updated: uui and progressdescription. They are changed if the value of the event is equal to the value of NCC_X_STATUS_INFO_UUI and/or the NCC_X_STATUS_INFO_PROGRESS descriptor.

Disconnecting function MyHangUp

MyHangUp is called whenever the program wants to hang up. The function first calls **nccGetCallStatus** to obtain the call's state. If the current call's state is NCC_CALLSTATE_DISCONNECTED (was the NCCEVN_CALL_DISCONNECTED event received), **nccReleaseCall** is called to release the call handle. Receipt of the NCCEVN_CALL_RELEASED event indicates that the call handle was cleared.

If the call state returned by **nccGetCallStatus** is different from NCC_CALLSTATE_DISCONNECTED, the function was called to initiate the disconnect process. **nccDisconnectCall** is called in this case, and **MyHangUp** waits for the NCCEVN_CALL_DISCONNECTED event indicating that the other party accepted the disconnect. **nccReleaseCall** is called to clear the call handle.

Switching function ConnectBChannel

ConnectBChannel does the following:

- It parses the NCC_ISDN_EXT_CALL_STATUS structure to find out the stream and timeslot on which the call has been placed (or received).
- It calls **swiMakeConection** to create a bi-directional connection between the physical B channel on the trunk and the appropriate DSP resource on the board.

10. ISDN TCP parameters

ISDN TCP parameters overview

This section describes the available parameters for ISDN Software call control. Parameter files are installed with NaturalAccess. The parameters in these files configure the TCP.

Parameter files are useful only if you are configuring the ISDNs Software in channelized configuration (as described in this manual). For more information, see Starting a TCP on a context.

Some parameters determine the amplitude, frequency, length, and pattern of the busy tone, ring tone, and reorder tone. Since the tones differ from country to country, these parameters are country-specific and must not be modified. Other parameters determine the service to request when placing an outbound call, the mode the TCP runs in, the channel direction, and the signal sending mask. These parameters can be modified for your application.

ISDN software parameter files

This topic provides an overview of the ISDN Software parameter files for the NCC API as well as information about Changing parameter values.

The following table describes the three types of parameter files that are installed for use with the NCC API:

File type / name	Description
*nccxadi**cty**.pf* *nccstart**cty**.pf* ***cty*** is the three character code of the target country. For example, the code for Australia is aus. The versions of these files for Australia are *nccxadiaus.pf* and *nccstartaus.pf*.	Binary parameter files containing a set of country-specific values for NCC API parameters. Do not change the values in these files. Changing certain values can affect the regulatory approvals in the target country.
nccxisdn.pf	A binary parameter file containing a set of ISDN Software parameters and default values. Changing these parameters directly affects all control messages (for example, messages associated with NCC API functions). To change them correctly, you must have knowledge of the ISDN specifications for the target country.
*nccxadi**cty**.par* *nccstart**cty**.par* *nccxisdn.par*	ASCII versions of *nccxadi**cty**.pf, nccstart**cty**.pf*, and *nccxisdn.pf.*

For NaturalAccess to load the binary parameter file, both of the binary parameter files (*.pf* files) for the target country must be in one of the directories specified with the AGLOAD environment variable. *nccxisdn.pf* must also be in the directory.

The NaturalAccess installation program asks you for a default country. It creates copies of the country-specific parameter files for that country, renames them, and places them in the AGLOAD path, as follows:

These files...	In this operating system...	Are copied to...
*nccxadi**cty**.pf* *nccxadi**cty**.par* where ***cty*** is the code for the default country	Windows	*\nms\ag\cfg\nccxadi.pf* *\nms\ag\cfg\nccxadi.par*
*nccxadi**cty**.pf* *nccxadi**cty**.par* where ***cty*** is the code for the default country	UNIX	*/opt/nms/ag/cfg/nccxadi.pf* */opt/nms/ag/cfg/nccxadi.par*
*nccstart**cty**.pf* *nccstart**cty**.par* *w*here ***cty*** is the code for the default country	Windows	*\nms\ag\cfg\nccstart.pf* *\nms\ag\cfg\nccstart.par*
*nccstart**cty**.pf* *nccstart**cty**.par* *w*here ***cty*** is the code for the default country	UNIX	*/opt/nms/ag/cfg/nccstart.pf* */opt/nms/ag/cfg/nccstart.par*
nccxisdn.pf *nccxisdn.par*	Windows	*\nms\ag\cfg\nccxisdn.pf* *\nms\ag\cfg\nccxisdn.par*
nccxisdn.pf *nccxisdn.par*	UNIX	*/opt/nms/ag/cfg/nccxisdn.pf* */opt/nms/ag/cfg/nccxisdn.par*

The files for only one country must appear in the AGLOAD directory. Otherwise, the parameters will not load.

Changing parameter values

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

Step	Action
1	Modify the value in the corresponding *.par* file.
2	Initialize the NaturalAccess PRM service so the binary parameter file is loaded.
3	Parse the *.par* file.

Step	Action
4	Do one of the following: • Call **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 system-wide parameters. See the *NaturalAccess Developer's Reference Manual* for more information. • Call **ctaLoadParameterFile** from within your application.

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

The following tables list all of the parameters in the *nccxidsn.pf* parameter file. Changing these parameters directly affects call control messages (for example, messages associated with NCC API functions). To change these parameters correctly, you must have knowledge of the ISDN specifications for the target country. See Using extended parameters for more information.

NCC.X.ADI_ISDN.ACCEPTCALL_EXT

Dependent function:

nccAcceptCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 accept cause.
progressdescriptor	0	integer	Q.931 PROGRESS description in PROGRESS message.

NCC.X.ADI_ISDN.ANSWERCALL_EXT

Dependent function:

nccAnswerCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).

NCC.X.ADI_ISDN.DISCONNECTCALL_EXT

Dependent function:

nccDisconnectCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 disconnect cause.
pad	0	integer	Pad.

NCC.X.ADI_ISDN.PLACECALL_EXT

Dependent function:

nccPlaceCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
CALLEDNUM.plan	0	integer	Q.931 numbering plan of called address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLEDNUM.type	0	integer	Q.931 numbering type of called address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.plan	0	integer	Q.931 numbering plan of calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.type	0	integer	Q.931 numbering type of calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.screen	0	integer	Q.931 screening indicator. (If supported. Possible values for this field are defined in *isdnval.h.*)

Field name	Default	Units	Description
CALLINGNUM.presentation	0	integer	Q.931 presentation indicator for calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
getcallid	0	integer	Set to 1 to request ***callid*** on call setup.
REDIRECTINGNUM.digits	""	char	The redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.pad	3	integer	Pad
REDIRECTINGNUM.plan	0	integer	Q.931 numbering plan of redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.type	0	integer	Q.931 numbering type of redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.screen	0	integer	Q.931 redirecting number screening indicator. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.presentation	0	integer	Q.931 presentation indicator for redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.reason	0	integer	Q.931 reason for redirection. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.pad1	0	integer	Pad1
service	0	integer	Q.931 service.
nsf_present	0	integer	Network-specific facilities (NSF): 1 = present 0 = not present

Field name	Default	Units	Description
nsf_service_feature	0	integer	NSF service or feature: 1 = SERVICE 0 = FEATURE
nsf_facility_coding	0	integer	NSF service or feature ID.

NCC.X.ADI_ISDN.REJECTCALL_EXT

Dependent function:

nccRejectCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 reject cause
pad	0	integer	Pad

NCC.X.ADI_ISDN.SENDDIGITS_EXT

Dependent function:

nccSendDigits

Field fame	Default	Units	Description
CALLEDNUM.plan	0	integer	Q.931 number plan of called address
CALLEDNUM.type	0	integer	Q.931 numbering type of called address

Possible values for these fields are defined in *isdnval.h.*

NCC.X.ADI_ISDN.START_EXT

Dependent function:

nccStartProtocol

Field name	Default	Units	Description
exclusive	1	integer	Determines if this TCP runs in exclusive mode or non-exclusive mode. For details, see B channel assignment overview. 0 - Non-exclusive mode 1 - Exclusive mode

Field name	Default	Units	Description
direction	0	integer	Determines if the channel is inbound, outbound, or bidirectional. See B channel assignment overview. 0 - bidirectional (default) 1 - inbound 2 - outbound
networkstream	0xFFFF	integer	Determines the stream to use for outbound calls. See B channel assignment overview.
networkslot	0xFFFF	integer	Determines the timeslot to use for outbound calls. See B channel assignment overview.
defaulttone	2	integer	Determines the tone to play when rejecting on timeout: 0 - reorder 1 - ringing 2 - busy
startCP	1	integer	If this parameter is set to 1, the TCP starts call progress when it receives a SETUP ACKNOWLEDGED, PROCEEDING, or ALERTING message, which contains a progress descriptor field or PROGRESS message, whichever comes first. If none of the listed messages arrive, call progress is started when CONNECTION CONFIRMATION is received, if the application does not want to connect on signal. If this parameter is set to 2, the TCP starts call progress on the first call establishment message (SETUP ACKNOWLEDGMENT, PROCEEDING, ALERTING, or PROGRESS) received back from the network, regardless of the progress descriptor. If this parameter is set to 0, call progress is not started.
flags	0x1121	mask	Flags (defined in *nccxisdn.h*) that determine when to send PROGRESS, CALL PROCEEDING, or ALERTING messages. The values can be ORed for cumulative effect. See Flags field bit settings.
blockrejectmode	0	integer	How to reject calls when the channel is blocked with BLOCK_REJECTALL mode: 0 - reject immediate 1 - play busy tone

Field name	Default	Units	Description
blockwaittime	3000	ms	Sets the maximum time to wait for the application to respond for an incoming call, before playing a default tone.
ISDNeventmask	0x0000	mask	ISDN informational event mask (defined in *nccxisdn.h*). ISDN_REPORT_PROGRESS indicates that ISDN progress messages are being received.

For more information about how parameter files are installed, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual*.

Flags field bit settings

Flags defined in *nccxisdn.h* determine when to send PROGRESS, CALL PROCEEDING, or ALERTING messages. The values can be ORed for cumulative effect.

This bit...	Causes protocol to send...
PROCEEDING_MASK	PROCEEDING on incoming call
PROGRESS_MASK_ANSWER	PROGRESS on **nccAnswerCall**
ALERTING_MASK_ANSWER	ALERT on **nccAnswerCall**
PROCEEDING_MASK_ANSWER	PROCEEDING on **nccAnswerCall**
PROGRESS_MASK_ACCEPT	PROGRESS on **nccAcceptCall**
ALERTING_MASK_ACCEPT	ALERT on **nccAcceptCall**
PROCEEDING_MASK_ACCEPT	PROCEEDING on **nccAcceptCall**
PROGRESS_MASK_REJECT	PROGRESS on **nccRejectCall**
ALERTING_MASK_REJECT	ALERT on **nccRejectCall**
PROCEEDING_MASK_REJECT	PROCEEDING on **nccRejectCall**

Changing parameter values

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

Step	Action
1	Modify the value in the corresponding *.par* file.
2	Initialize the NaturalAccess PRM service so the binary parameter file is loaded.
3	Parse the *.par* file.
4	Do one of the following: • Call **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 system-wide parameters. See the *Dialogic® NaturalAccess™ Software Developer's Manual* for more information. • Call **ctaLoadParameterFile** from within your application.

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

The following tables list the parameters in the *nccxidsn.pf* parameter file. Changing these parameters directly affects call control messages (for example, messages associated with NCC API functions). To change these parameters correctly, you must have knowledge of the ISDN specifications for the target country. See Using extended parameters for more information.

NCC.X.ADI_ISDN.ACCEPTCALL_EXT

Dependent function:

nccAcceptCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 accept cause.
progressdescriptor	0	integer	Q.931 PROGRESS description in PROGRESS message.

NCC.X.ADI_ISDN.ANSWERCALL_EXT

Dependent function:

nccAnswerCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).

NCC.X.ADI_ISDN.DISCONNECTCALL_EXT

Dependent function:

nccDisconnectCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 disconnect cause.
pad	0	integer	Pad.

NCC.X.ADI_ISDN.PLACECALL_EXT

Dependent function:

nccPlaceCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
CALLEDNUM.plan	0	integer	Q.931 numbering plan of called address. (If supported. Possible values for this field are defined in *isdnval.h*.)
CALLEDNUM.type	0	integer	Q.931 numbering type of called address. (If supported. Possible values for this field are defined in *isdnval.h*.)

Field name	Default	Units	Description
CALLINGNUM.plan	0	integer	Q.931 numbering plan of calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.type	0	integer	Q.931 numbering type of calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.screen	0	integer	Q.931 screening indicator. (If supported. Possible values for this field are defined in *isdnval.h.*)
CALLINGNUM.presentation	0	integer	Q.931 presentation indicator for calling address. (If supported. Possible values for this field are defined in *isdnval.h.*)
getcallid	0	integer	Set to 1 to request ***callid*** on call setup.
REDIRECTINGNUM.digits	""	char	The redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.pad	3	integer	Pad
REDIRECTINGNUM.plan	0	integer	Q.931 numbering plan of redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.type	0	integer	Q.931 numbering type of redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.screen	0	integer	Q.931 redirecting number screening indicator. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.presentation	0	integer	Q.931 presentation indicator for redirecting address. (If supported. Possible values for this field are defined in *isdnval.h.*)

Field name	Default	Units	Description
REDIRECTINGNUM.reason	0	integer	Q.931 reason for redirection. (If supported. Possible values for this field are defined in *isdnval.h.*)
REDIRECTINGNUM.pad1	0	integer	Pad1
service	0	integer	Q.931 service.
nsf_present	0	integer	Network-specific facilities (NSF): 1 = present 0 = not present
nsf_service_feature	0	integer	NSF service or feature: 1 = SERVICE 0 = FEATURE
nsf_facility_coding	0	integer	NSF service or feature ID.

NCC.X.ADI_ISDN.REJECTCALL_EXT

Dependent function:

nccRejectCall

Field name	Default	Units	Description
uui	""	char	User-to-user information (up to 132 characters).
cause	0	integer	Q.931 reject cause
pad	0	integer	Pad

NCC.X.ADI_ISDN.SENDDIGITS_EXT

Dependent function:

nccSendDigits

Field fame	Default	Units	Description
CALLEDNUM.plan	0	integer	Q.931 number plan of called address
CALLEDNUM.type	0	integer	Q.931 numbering type of called address

Note: Possible values for these fields are defined in *isdnval.h.*

NCC.X.ADI_ISDN.START_EXT

Dependent function:

nccStartProtocol

Field name	Default	Units	Description
exclusive	1	integer	Determines if this TCP runs in exclusive mode or non-exclusive mode. For details, see B channel assignment overview. 0 - Non-exclusive mode 1 - Exclusive mode
direction	0	integer	Determines if the channel is inbound, outbound, or bidirectional. See B channel assignment overview. 0 - bidirectional (default) 1 - inbound 2 - outbound
networkstream	0xFFFF	integer	Determines the stream to use for outbound calls. See B channel assignment overview.
networkslot	0xFFFF	integer	Determines the timeslot to use for outbound calls. See B channel assignment overview.
defaulttone	2	integer	Determines the tone to play when rejecting on timeout: 0 - reorder 1 - ringing 2 - busy
startCP	1	integer	If this parameter is set to 1, the TCP starts call progress when it receives a SETUP ACKNOWLEDGED, PROCEEDING, or ALERTING message, which contains a progress descriptor field or PROGRESS message, whichever comes first. If none of the listed messages arrive, call progress is started when CONNECTION CONFIRMATION is received, if the application does not want to connect on signal. If this parameter is set to 2, the TCP starts call progress on the first call establishment message (SETUP ACKNOWLEDGMENT, PROCEEDING, ALERTING, or PROGRESS) received back from the network, regardless of the progress descriptor. If this parameter is set to 0, call progress is not started.

Field name	Default	Units	Description
flags	0x1121	mask	Flags (defined in *nccxisdn.h*) that determine when to send PROGRESS, CALL PROCEEDING, or ALERTING messages. The values can be ORed for cumulative effect. See Flags field settings.
blockrejectmode	0	integer	How to reject calls when the channel is blocked with BLOCK_REJECTALL mode: 0 - reject immediate 1 - play busy tone
blockwaittime	3000	ms	Sets the maximum time to wait for the application to respond for an incoming call, before playing a default tone.
ISDNeventmask	0x0000	mask	ISDN informational event mask (defined in *nccxisdn.h*). ISDN_REPORT_PROGRESS indicates that ISDN progress messages are being received.

For more information about how parameter files are installed, see the *Dialogic® NaturalAccess™ ISDN Software Installation Manual.*

Flags field settings

Flags (defined in *nccxisdn.h*) that determine when to send PROGRESS, CALL PROCEEDING, or ALERTING messages. The values can be ORed for cumulative effect.

This bit...	Causes protocol to send...
PROCEEDING_MASK	PROCEEDING on incoming call
PROGRESS_MASK_ANSWER	PROGRESS on **nccAnswerCall**
ALERTING_MASK_ANSWER	ALERT on **nccAnswerCall**
PROCEEDING_MASK_ANSWER	PROCEEDING on **nccAnswerCall**
PROGRESS_MASK_ACCEPT	PROGRESS on **nccAcceptCall**
ALERTING_MASK_ACCEPT	ALERT on **nccAcceptCall**
PROCEEDING_MASK_ACCEPT	PROCEEDING on **nccAcceptCall**
PROGRESS_MASK_REJECT	PROGRESS on **nccRejectCall**
ALERTING_MASK_REJECT	ALERT on **nccRejectCall**
PROCEEDING_MASK_REJECT	PROCEEDING on **nccRejectCall**

11. Index