Dialogic
Making Innovation Thrive™

Dialogic® NaturalAccess™ Point-to-Point Switching API Developer's Manual

October 2009 64-0503-01

www.dialogic.com

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
9000-60007-10	July 2000	LBG, Platform Support for Fusion 4.0
9000-60007-11	September 2000	LBG, CT Access 4.0
9000-60007-12	March 2001	GJG, NACD 2000-2
9000-60007-13	November 2001	MCM, NACD 2002-1 Beta
9000-60007-14	May 2002	SRG, NACD 2002-1
9000-60007-15	April 2003	SRR, NACD 2003-1
9000-60007-16	April 2004	MCM, Natural Access 2004-1
9000-60007-17	March 2005	LBG, Natural Access 2005-1
64-0503-01	October 2009	LBG, NaturalAccess R9.0
Last modified: September 4, 2009		

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

Table Of Contents

1 Introduction

The *Dialogic® NaturalAccess™ Point-to-Point Switching API Developer's Manual* provides:

- An overview of the Point-to-Point Switching (PPX) service.
- A reference of its functions and errors.

This manual defines telephony terms where applicable, but assumes that you are familiar with telephony and switching concepts, and with the C programming language.

Read the *Dialogic® NaturalAccess™ Software Developer's Manual* before developing a Natural Access switching application. The *Dialogic® NaturalAccess™ Software Developer's Manual* contains detailed information about Natural Access concepts, architecture, and application development.

2 Terminology

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

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

Former terminology	Dialogic terminology
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
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

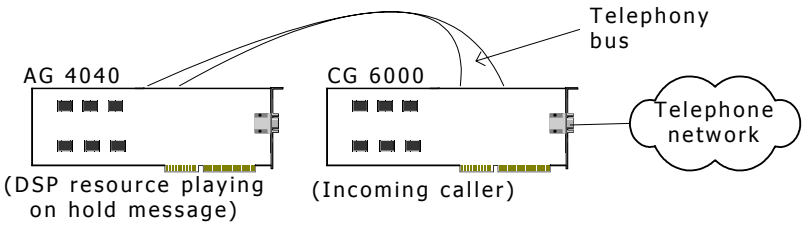
3 Overview of the Point-to-Point Switching service

Point-to-point switching service definition

The Point-to-Point Switching (PPX) service is a Natural Access service that enables you to make switch connections between boards connected by a telephony bus without having to specify the intervening bus timeslots. The PPX service maintains an internal database that records the switching configuration for the system and the state of all the timeslots.

To make a switch connection using PPX, specify the two endpoints for the connection. The PPX service manages the telephony bus timeslots and controls the underlying hardware switches to make the connections. The PPX service can be used with any board that has an FMIC or HMIC type switch.

In the following example, a connection is required between a DSP resource on an AG 4040 board and a T1 channel on an CG 6000 board. The DSP resource on the AG 4040 board is used to play an on hold message for all incoming calls.



Using the PPX service, the application calls one function specifying both the location of the DSP resource (board number, local stream, and timeslot) and the location of the caller (board number, local stream, and timeslot). The PPX service determines the required telephony bus timeslots and makes the necessary switch connections.

In contrast, when using Natural Access Switching (SWI) service functions to make the connection, an application is required to:

1. Determine if the switch connection can use the local bus or if it requires telephony bus timeslots.
2. Determine which telephony bus timeslots are available for the connection.
3. Send the switch commands to connect the caller to an available timeslot on the telephony bus.
4. Send the switch commands to connect the DSP resources to the telephony bus.

When using the PPX service, all streams and timeslots are addressed using the MVIP-95 switch model. The Switching service provides additional functions for configuring telephony bus clocks and stream speed. Use these commands to configure the bus before making any connections. Refer to the *Switching Service Developer's Reference Manual* for more information.

Caution:	When using the PPX service, do not use the SWI service to make switch connections. The PPX service maintains an internal connection database to determine local resource and telephony bus usage. Since connections made with the SWI service are not recorded in this database, you can have unpredictable results. The PPX service could use timeslots already used by the SWI service and the SWI service could use timeslots already used by the PPX service.

PPX terminology

An application can use the PPX service to:

- Manage connections.
- Make one-talker, one-listener switch connections.
- Perform actions on groups of connections in a single transaction.

The following table defines the basic elements of the PPX service:

Term	Definition
Connection object	An internal data structure maintained by the PPX server that is created when a switch connection is successfully opened. This structure records the state of the connection. In the PPX service, a connection object can be considered as an empty container until talkers and listeners are assigned to it.
Connection handle	An identifier for a specific connection object.
Connection	A logical path between two terminus entities. Within the PPX service, a connection can have only one talker but can support multiple listeners.
Terminus	A single access point to a switch block input or switch block output. A terminus contains a bus, a stream, a switch number, and a timeslot.
Talker	The terminus that drives the output on the connection.
Listener	The terminus that receives input from the connection.

Properties of connections and listeners

PPX connections and PPX listeners have the following unique properties:

- Switch connections established through the PPX service have an ownership property.

  When a PPX client creates a connection between two points, only the client that established the connection (who owns it) is allowed to break the connection (which disables output).
- Ownership can be overridden by assigning the connection a name, allowing different contexts bound to different clients to operate on the same connection points.
- PPX listener points have an associated state value which is either BUSY or NOT_BUSY.

Listener points that are connected to a talker are in the BUSY state. A listener point can be connected to a talker only when that point is in the NOT_BUSY state. Therefore, if talker point A is connected to listener point B, listener point B must be disconnected from talker A before it can be assigned to a new talker point C.

Setting up the Natural Access environment

Before calling PPX functions, the application must set up the Natural Access environment by performing the following steps:

Step	Action
1	Initialize the Natural Access application.
2	Create event queues.
3	Create contexts and attach them to event queues.
4	Open services on each context.

Complete the following steps to set up a second Natural Access application that shares a context with the first application:

Step	Action
1	Initialize the Natural Access application.
2	Create event queues.
3	Attach the application to the existing context.

Initializing Natural Access

Register services in the call to **ctaInitialize** by specifying the service name (PPX) and the service manager name (PPXMGR). Only the services initialized in the call to **ctaInitialize** can be opened by the application.

Creating event queues and contexts

After initializing Natural Access, create the event queues and contexts. An event queue is the communication path from a Natural Access service to an application. A Natural Access service generates events indicating certain conditions or state changes. An application retrieves the events from the event queue. Most PPX functions are synchronous, returning when the function is complete and not generating events.

Create one or more event queues by calling **ctaCreateQueue**. Specify which service managers is attached to each queue. When you attach or bind a service manager to an event queue, you make that service manager available to the event queue.

Natural Access organizes services and accompanying resources around a single processing context. A context usually represents an application instance controlling a single telephone call. Natural Access provides multi-processing support: multiple Natural Access application processes can perform tasks on behalf of the same context (referred to as context sharing). Natural Access applications can transfer control of contexts (for example, contexts associated with individual telephone calls) to other Natural Access applications (referred to as context hand-off).

Create a context by calling **ctaCreateContext** or **ctaCreateContextEx**. Provide the queue handle (***ctaqueuehd***) that was returned from **ctaCreateQueue**. All events for services on the context are received in the specified queue. **ctaCreateContext** returns a context handle (***ctahd***).

Opening services

To open services on a context, call **ctaOpenServices** and pass a context handle and a list of service descriptors. The service descriptor specifies the name of the service and the service manager.

Note: Open the SWI service in a PPX client application only if the application is making calls directly to the SWI service on its own behalf. For example, if a PPX client application sets the telephony bus clocks and stream speeds, it must use SWI functions and must open the SWI service.

Refer to the *Natural Access Developer's Reference Manual* for details on initializing Natural Access and for programming models you can use.

PPX service deliverables

The following table lists the PPX service deliverables:

File	Description
Windows: *\nms\bin\ppxapi.dll* UNIX: */opt/nms/bin/libppxapi.so*	PPX service interface library
Windows: *\nms\bin\ppxmgr.dll* UNIX: */opt/nms/bin/libppxmgr.so*	PPX service manager library
Windows: *\nms\bin\ppxserv.exe* UNIX: */opt/nms/bin/ppxserv*	PPX server
Windows: *\nms\bin\ppxservicecfg.exe*	Utility to install the PPX server as a Windows service
Windows: *\nms\bin\ppxstop.exe* UNIX: */opt/nms/bin/ppxstop*	Utility to shutdown the PPX server
Windows: *\nms\ctaccess\cfg\ppx_tmpl.cfg* UNIX: */opt/nms/ctaccess/cfg/ppx_tmpl.cfg*	PPX service example configuration file
Windows: *\nms\ctaccess\demos\brd2brd\brd2brd.exe* UNIX: */opt/nms/ctaccess/demos/brd2brd/brd2brd*	Demonstration program that performs call transfer from an incoming line to an outgoing line over the MVIP bus using the PPX service.
Windows: *\nms\ctaccess\demos\brd2brd\brd2brd.c* UNIX: */opt/nms/ctaccess/demos/brd2brd/brd2brd.c*	Demonstration program source code
Windows: *\nms\ctaccess\demos\brd2brd\makefile* UNIX: */opt/nms/ctaccess/demos/brd2brd/makefile*	Makefile for compiling the demonstration program

File	Description
Windows: *\nms\ctaccess\demos\callcntr\callcntr.exe* UNIX: */opt/nms/ctaccess/demos/callcntr/callcntr*	Demonstration program that performs call transfer from an incoming line to an outgoing line over the CT bus using the PPX service connection objects
Windows: *\nms\ctaccess\demos\callcntr\callcntr.c* UNIX: */opt/nms/ctaccess/demos/callcntr/callcntr.c*	Demonstration program source code
Windows: *\nms\ctaccess\demos\callcntr\makefile* UNIX: */opt/nms/ctaccess/demos/callcntr/makefile*	Makefile for compiling the demonstration program
Windows: *\nms\include\ppxdef.h* UNIX: */opt/nms/include/ppxdef.h*	Header file for the PPX service
Windows: *\nms\lib\ppxapi.lib*	PPX service interface import library
Windows: *\nms\lib\ppxmgr.lib*	PPX service manager import library

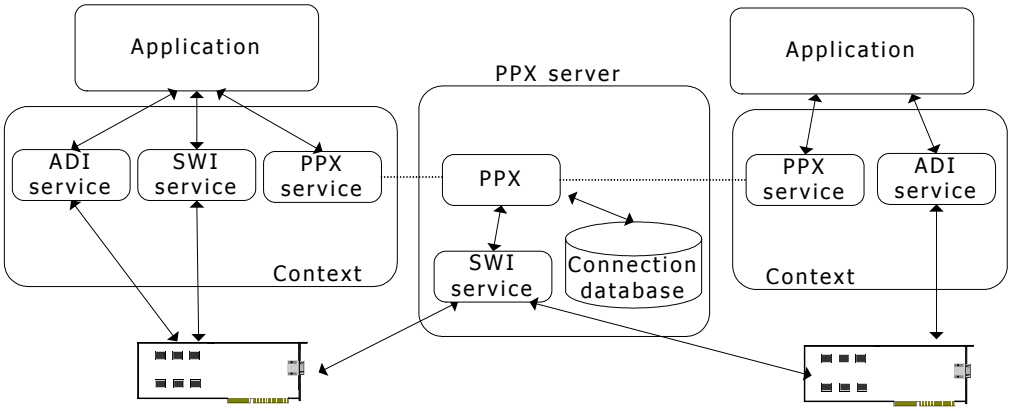
Note: Applications must link with the service interface library.

PPX server

Since the same telephony bus can be used in separate Natural Access applications, the PPX service must be able to manage the bus streams and timeslots across multiple applications. Each instance of the PPX service that is opened on a context runs as a client to a PPX server. The PPX server (*ppxserv*) manages an internal connection database to determine which timeslots are available on the telephony bus and what connections have been made in the system.

As shown in the following illustration, each application creates a context containing the PPX service, the ADI service, and the SWI service. Opening the PPX service on the context creates an instance of a client to the PPX server. Each PPX client can be a separate process, but it is not required. A unique PPX client can be associated with each thread running in a single process.

All commands sent by the applications to the PPX service are sent by the clients to the PPX server. The PPX server sends the commands to the board using the SWI service functions. Each application does not have to open the SWI service on a context. The PPX server automatically opens an instance of the SWI service.



Before you start up the PPX server:

- Start the Natural Access daemon (*ctdaemon*).
- Ensure that the boards are booted (run *oamsys*).
- Ensure that the system switching topology (switch fabric) is defined in the PPX configuration file (*ppx.cfg*). Refer to *Switch configuration overview* on page 35 for details on creating *ppx.cfg*.

The PPX server must be running while any application using the PPX service is running.

This topic presents information on:

- Using the PPX server on Windows
- Using the PPX server on UNIX
- PPX initialization error logging
- Switching service error logging

Using the PPX server on Windows

During system development and debugging of *ppx.cfg*, control the PPX server manually so it can be restarted as needed.

To start the PPX server from the command line, enter:

```
start ppxserv [options]
```

where ***options*** are:

Option	Description	Default
-f ***filename***	Specifies the configuration file name.	*\nms\ctaccess\cfg\ppx.cfg*
-e ***filename***	Specifies the error log file name.	*\nms\ctaccess\pxerror.log*
-d ***filename***	Specifies the database output file name.	*\nms\ctaccess\ppxdb.log*
-l	Turns on the SWI command logging process.	OFF
-w ***number***	Specifies the line number at which the SWI log file will wrap to start.	2000
-s ***filename***	Specifies the SWI command log file name.	*\nms\ctaccess\ppxswi.log*

To stop the PPX server from the command line, enter:

```
ppxstop
```

Using the *ppxstop* command allows the server to terminate any connections it has established and to release communication resources.

Using the PPX server as a Windows service

Once *ppx.cfg* is verified, install the PPX server as a Windows service using the *ppxservicecfg* utility.

Once the PPX server is installed in the Windows registry, run it as follows:

Step	Action
1	Select **Services** from the Control Panel.
2	Highlight **Point-to-Point Telephony Switching**.
3	Select **Start** or **Stop** to manually start or stop the PPX server.
4	Select **Startup**/**Automatic** as the startup option to automatically start the PPX server when the system reboots.

Starting and stopping the PPX server

You can control the PPX server from the command line after you install it as a Windows service. Before stopping the PPX server, shut down all client applications to break all switch connections created by the PPX clients.

To start the PPX server from the command line enter:

```
net start ppx
```

To stop the PPX server from the command line enter:

```
ppxstop
```

Removing the PPX server from Windows services

To remove the PPX server from Windows services and reset the Windows registry, enter:

```
ppxservicecfg -remove
```

Using the PPX server on UNIX

Before starting the PPX server, make sure a valid *ppx.cfg* exists.

Complete the following steps to start the PPX server:

Step	Action
1	Log on as root.
2	Enter the following command: `/opt/nms/bin/ppxserv [options]` where ***options*** are:

Option	Description	Default
-b	Forces the PPX server to run as a background process. Used instead of & to ensure that the server detaches from the console window at initialization.	
-f ***filename***	Specifies the configuration file name.	*/var/opt/nms/ctaccess/cfg/ppx.cfg*
-e ***filename***	Specifies the error log file name.	*/var/opt/nms/ctaccess/ppxerror.log*
-d ***filename***	Specifies the database output file name.	*/var/opt/nms/ctaccess/ppxdb.log*
-l	Turns on the SWI command logging process.	OFF
-w ***number***	Specifies the line number at which the SWI log file will wrap to start.	2000
-s ***filename***	Specifies the SWI command log file name.	*/var/opt/nms/ctaccess/ppxswi.log*

Stopping the PPX server

Before stopping the PPX server, shut down all client applications so that all switch connections created by the PPX clients are automatically broken by the PPX server.

Run the *ppxstop* utility to terminate any connections the server has established and to release communication resources.

PPX initialization error logging

When initialized, the PPX server creates an error log file that records initialization errors and problems associated with the interpretation of *ppx.cfg*. By default, this error file is located as follows:

Operating system	Directory
Windows	*\nms\ctaccess\ppxerror.log*
UNIX	*/var/opt/nms/ctaccess/ppxerror.log*

If the PPX server terminates abnormally, all switch connections it established at that time remain in place. Restarting the PPX server does not clear these underlying switch connections. Use the *swish* utility to reset the switches.

For example, for an AG 2000 board, start up *swish* and enter the following command:

```
openswitch  s0 = agsw 0
resetswitch s0
```

Changing the communications port number

Natural Access applications use the Point-to-Point Switching service to communicate with the PPX server. The communications mechanism is socket based, and the default port number is 1110. If you see PPXERR_COMM_FAILURE in the PPX error log when you start up your Natural Access application, change the port number to an unused port number.

Complete the following steps to change the port number:

Step	Action
1	Review your system configuration and determine a new port number to use.
2	Set the NMS_PPX_COMM environment variable to the new port number by using the appropriate mechanism for your operating system and shell.
3	Restart the PPX server and all applications.

SWI error logging

The Point-to-Point Switching service is a layer of software built on top of the Switching (SWI) service. It uses the following SWI service functions:

- **swiOpenSwitch**
- **swiMakeConnection**
- **swiDisableOutput**
- **swiSendPattern**
- **swiCloseSwitch**

The PPX service can log all SWI service function calls to a PPX log file if you pass an optional parameter (-l) when you start the PPX server. The log includes an error indicator (*) when necessary. The actual value from the SWI service is at the end of the line.

The PPX/SWI log file is located and named as follows:

Operating system	Directory
Windows	*\nms\ctaccess\ppxswi.log*
UNIX	*/var/opt/nms/ctaccess/ppxswi.log*

The log file is circular, which prevents excessive use of disk space. By default, the log file contains the last 2000 SWI service calls made by the PPX server. The SWI service calls are time stamped so you can track starting and ending points.

To start the PPX server and enable SWI service logging, start the PPX server from the command line with the argument -l.

To change the default number of SWI log entries stored in the log, start the PPX server from the command line with the following arguments:

```
-l -w new_value
```

For example:

To restrict SWI logging to 10,000 entries on a Windows system, enter the following command:

```
start \nms\bin\ppxserv -l -w 10000
```

To perform the same modification on a UNIX system:

Step	Action
1	Log on as root.
2	Start the PPX server in the background by entering: `/opt/nms/bin/ppxserv -l -w 10000 -b`

Note: When you restart the PPX server with logging enabled, it overwrites any existing PPX/SWI log files on the system. If you need to preserve your data, save the log file under a different file name before you restart the PPX server.

A sample ppxswi.log

This section provides an excerpt from a *ppxswi.log* file. The PPX server internally shares open switch handles among all client instances.

Note: The ***ret:*** value that appears on the line marked by an asterisk is the error return value from the Switching service.

Because the log file is circular, the date stamp and the time stamp of each issued command are recorded so you can identify the most recent entries after the file reaches its maximum capacity.

```
[Fri Jan 22 10:00:20 1999] Switch 0: Handle Opened 0x1
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:0:2 TO LOCAL:5:2
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:4:2 TO LOCAL:1:2
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:2:2 TO LOCAL:7:2
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:6:2 TO LOCAL:3:2
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:0:4 TO LOCAL:5:4
[Fri Jan 22 10:00:20 1999]   swiMakeConnect: hd 0x1 LOCAL:4:4 TO LOCAL:1:4
[Fri Jan 22 10:02:44 1999]   swiMakeConnect: hd 0x1 LOCAL:0:6 TO LOCAL:5:6
[Fri Jan 22 10:03:09 1999] Switch 1: Handle Opened 0x10002
[Fri Jan 22 10:03:09 1999]   swiMakeConnect: hd 0x10002 CTBUS:0:31 TO LOCAL:1:6
[Fri Jan 22 10:03:09 1999]   swiMakeConnect: hd 0x1 LOCAL:0:6 TO CTBUS:0:31
[Fri Jan 22 10:03:14 1999]   swiMakeConnect: hd 0x10002 CTBUS:0:31 TO LOCAL:1:4
[Fri Jan 22 10:07:47 1999]   swiMakeConnect: hd 0x10002 LOCAL:0:0 TO CTBUS:5:0
[Fri Jan 22 10:53:24 1999]   swiDisableOutput: hd 0x10002 CTBUS:5:0
[Fri Jan 22 10:54:03 1999] * swiMakeConnect: hd 0x1 LOCAL:8:0 TO CTBUS:8:0
                             ret: 0x40004
```

4 Developing applications

Managing PPX connections

The fundamental element in the Point-to-Point Switching service is a connection object. A connection is basically a telephone call. Each connection can have one talker. The talker in the connection is driving the output for that connection. Each connection can have one or more listeners. The listeners receive input from the connection.

Use the PPX service to perform the following operations:

- Create a connection
- Set the talker for a connection
- Add listeners to a connection
- Remove listeners from a connection
- Set the silence pattern for a connection
- Open a connection
- Close a connection
- Destroy a named connection

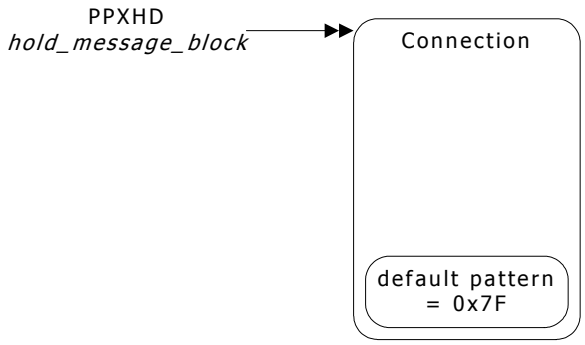
Creating a connection

Create a connection by calling **ppxCreateConnection** with a context. A handle (***ppxhd***) to the connection is returned. The connection handle is used to specify the connection when setting a talker or adding listeners to the connection. When a connection is initially created, there are no telephony bus timeslots associated with it.

A default idle bit pattern (a fixed 8-bit pattern) can be specified in the call to **ppxCreateConnection**. If there is no talker attached, any listeners added to the connection receive the default pattern sent on the stream attached to the listener.

In the following topics, an example application is used to demonstrate the PPX functions. The example application runs a system in which incoming callers are placed on hold while waiting for an available operator. All callers that are on hold listen to the same voice message.

The following illustration shows the creation of the connection, ***hold_message_block***. A default pattern is set and a pointer to the connection is returned.



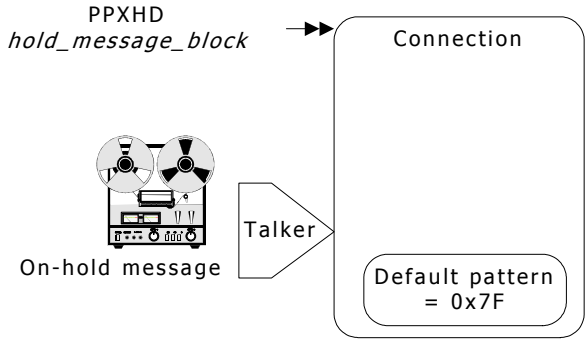
When creating a connection, you can assign a name to the connection. A named connection can be accessed by other applications or contexts (as discussed in Opening a connection), allowing the ownership property to be over-ridden.

Setting the talker for a connection

ppxSetTalker sets a talker for a connection. Specify the address of the talker by the switch identification number, bus type (CTBUS or LOCAL), stream, and timeslot.

If listeners are already attached to the connection, the PPX service connects the listeners to the talker using underlying switch connections. The switch connections can involve the allocation of bus timeslots if the talkers and the listeners are on different boards. If there is a talker already attached to the connection, the original talker is automatically disconnected before the new talker is connected.

In the following illustration, the talker is added to the ***hold_message_block***. The talker is a DSP resource playing the on-hold message.

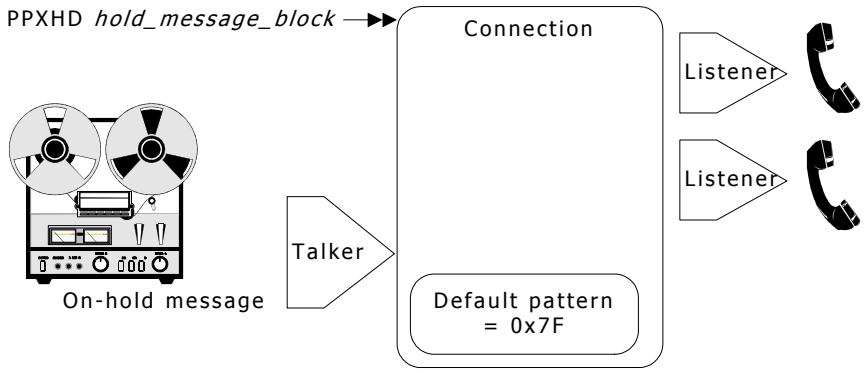


Adding listeners to a connection

ppxAddListeners adds listeners to a connection. Specify the listeners by an array containing the switch identification number, bus type (CTBUS or LOCAL), stream, and timeslot for each listener.

If the connection already has a talker attached, the listeners are connected to the talker using switch connections. The switch connections can involve the allocation of bus timeslots if the talker and the listener are on different boards. If there is no talker attached to the connection, the listeners have the default pattern of the connection sent to them.

In the following illustration, as incoming calls are received, they are connected to the ***hold_message_block*** to receive the on-hold message while waiting for an available operator.

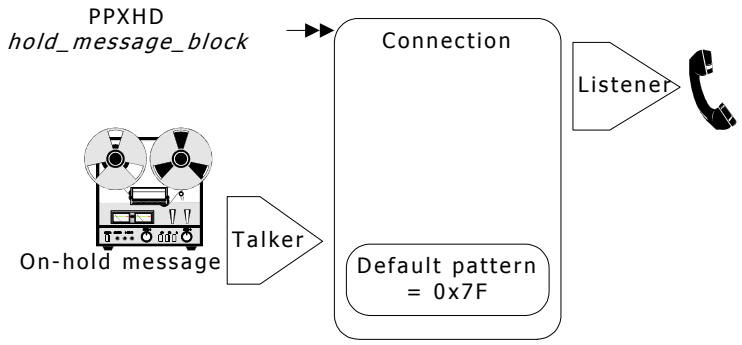


Removing listeners from a connection

ppxRemoveListeners removes listeners from a connection. Specify the listeners by an array containing the switch identification number, bus type (CTBUS or LOCAL), stream, and timeslot for each listener.

If the listeners are connected to a talker, the switch connections are broken. The listeners are put into their default disabled hardware state. For telephony bus output streams, the default hardware state is a high impedance state. Local bus output streams are put into the vendor-specified default state.

As shown in the following illustration, as an operator becomes available, listeners are removed from the ***hold_message_block*** so they can be connected to an operator.



Setting the silence pattern for a connection

ppxSetDefaultPattern sets the default pattern for a connection. If there is no talker attached to a connection, the listeners receive the default pattern. If listeners are connected to a talker and the talker is removed, the listeners receive the default pattern for the connection.

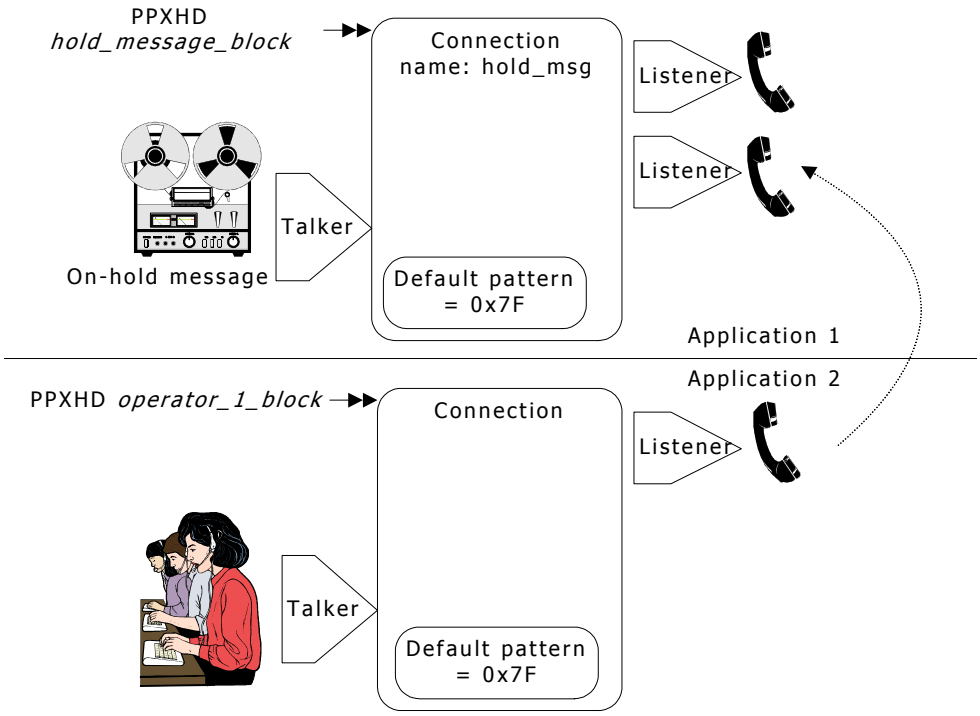
This function overrides the default pattern set when **ppxCreateConnection** is called to create the connection. The default silence pattern used by the server is 0x7f (mu-law). This default value can be modified by setting a switch fabric attribute within the PPX configuration file, as described in the *Switch configuration overview* on page 35.

Opening a connection

ppxOpenConnection returns a handle to a named connection.

Typically in Natural Access, each application runs as a separate thread or process. If a connection is assigned a name when it is created, each application can open a handle to that connection and add a talker, add listeners, or remove listeners. Any action taken in one application is reflected in the connection and is apparent in every application that accesses that connection.

Note: The PPX service considers a PPX client to be equivalent to a Natural Access context.



In the previous illustration, the ***hold_message_block*** was created with the name hold_msg. In the system, one application is managing incoming calls and transferring them to operators when they are available. Another application is used to manage the operator stations. If an operator needs to place the caller back on-hold, the application can open the connection named hold_msg. The connection handle ***hold_message_block*** is returned and the application can then add the caller as a listener to that connection.

Closing a connection

ppxCloseConnection closes a previously created connection. If the connection is unnamed (refer to *Creating a connection* on page 21 for information about naming connections), the resources associated with the connection are freed and all connection outputs become disabled. A named connection is closed by **ppxDestroyNamedConnection**.

Destroying a named connection

ppxDestroyNamedConnection frees all resources and disables outputs to all points on a named connection. Even if all applications close a named connection with **ppxCloseConnection**, the connection remains in the PPX connection database until it is explicitly destroyed with **ppxDestroyNamedConnection**.

Once destroyed, all open handles to the named connection become invalid in all applications.

Making one-talker, one-listener connections

The PPX service provides convenience functions, **ppxConnect** and **ppxDisconnect**, to make connections between one talker and one listener.

ppxConnect uses internal connection handles that cannot be accessed by the application to make connections. Additional listeners cannot be these connections. These connections cannot be accessed across applications.

When a connection is made using **ppxConnect**, the connection is owned by the context that created it. Only the owner context can disconnect the connection by calling **ppxDisconnect**.

These convenience functions include a ***mode*** argument that directs the PPX service to calculate endpoints for any additional connections based upon the single talker and listener that have been passed. The ***mode*** argument can have any of the following values:

Value	Description
PPX_SIMPLEX	Establishes a connection in one direction. This mode is typically used to broadcast or to monitor half a conversation.
PPX_DUPLEX	Establishes connections in two directions. This mode is typically used to connect a phone conversation.
PPX_QUAD	Establishes two duplex connections, one for voice and one for signaling. This mode is typically used to connect a network interface to its voice and signaling DSPs.

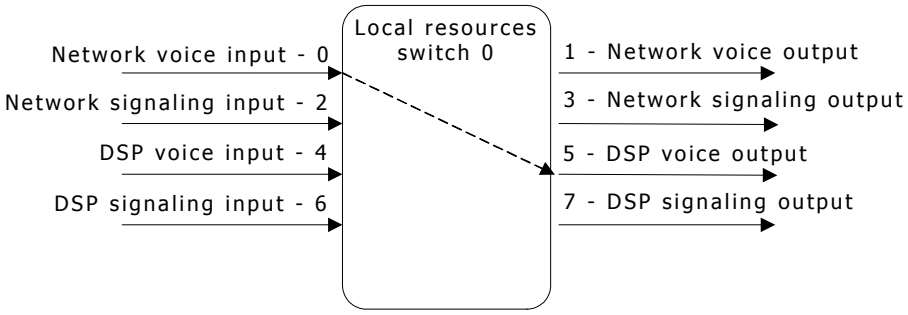
This topic provides information on:

- Making a simplex connection
- Making a duplex connection
- Making a quad connection
- Disconnecting a talker and a listener

Making a simplex connection

To make a simplex connection, call **ppxConnect**, specify the address of the talker and the listener, and specify the mode PPX_SIMPLEX. The PPX service connects the talker to the listener by making the necessary switch connection. If the talker and the listener are on different boards, the connection can involve the allocation of telephony bus timeslots.

The following illustration shows a simplex connection between local resources on a telephony board. The switch connection connects a network interface voice input on local stream 0 to the DSP voice output on local stream 5.



talker = 0:LOCAL:0:3
listener = 0:LOCAL:5:3

ppxConnect (***ctahd***, ***talker***, ***listener***, PPX_SIMPLEX)
LOCAL:0:3 -> LOCAL:5:3

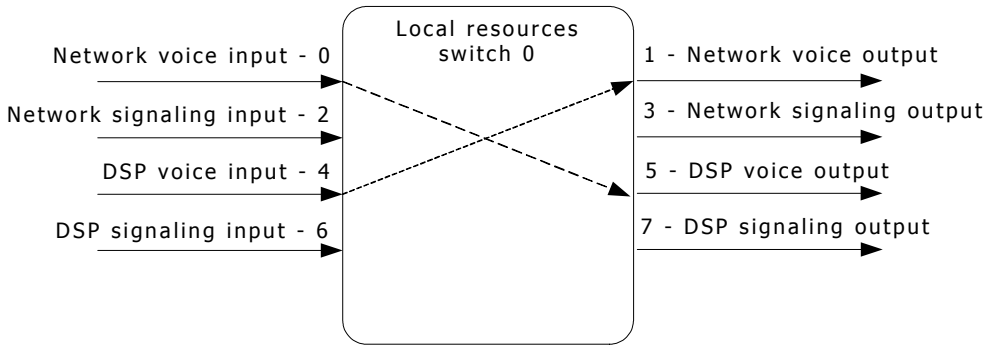
Making a duplex connection

To make a duplex connection, call **ppxConnect**, specify the address of the talker and the listener, and specify the mode PPX_DUPLEX. For duplex connections, the PPX service makes two end-to-end connections. If the talker and the listener are on different boards, the connections involve the allocation of telephony bus timeslots.

A duplex connection is a bidirectional voice or a bidirectional signaling connection. Telephony bus streams are typically defined in pairs; an even-numbered stream ***n*** has a corresponding stream ***n+1***. The even-numbered stream is in one direction and the odd-numbered stream is in the other direction. For example, if stream 0 is an input stream, stream 1 is an output stream.

For a duplex PPX connection, the talker and listener that are passed as arguments to the function specify the first connection. The second connection is determined by connecting the stream pair.

The following illustration shows a duplex connection between local resources on a telephony board. **ppxConnect** is called with the address of the talker (LOCAL:0:3) and the address of the listener (LOCAL:5:3). The first connection makes the connection between the talker and the listener. The PPX service automatically makes the second connection that is LOCAL:4:3 (stream 4 is the pair of stream 5) to LOCAL:1:3 (stream 1 is the pair of stream 0).



talker = 0:LOCAL:0:3
listener = 0:LOCAL:5:3

ppxConnect (***ctahd***, ***talker***, ***listener***, PPX_DUPLEX)
LOCAL:0:3 -> LOCAL:5:3
LOCAL:4:3 -> LOCAL:1:3

The PPX service uses the following algorithm to determine the streams that are used in a duplex connection. The application specifies one talker and one listener in the command:

```
ppxConnect( ctahd, &talker, &listener, PPX_DUPLEX)
```

The PPX service makes a series of connections. The first connection is:

```
talker -> listener
```

The second connection depends on whether the specified bus is local. Since inputs from local resources are always on even-numbered streams and outputs to local resources are always on odd-numbered streams, local stream endpoints are always as follows:

```
listener_2.stream = talker.stream + 1
talker_2.stream = listener.stream - 1
```

Then connect:

```
talker_2 -> listener_2
```

When the bus is not local, the second connection is determined as follows:

- If the talker is on an even-numbered stream:

  ```
  listener_2.stream = talker.stream + 1
  ```

  otherwise:

  ```
  listener_2.stream = talker.stream - 1
  ```

- If the listener is on an even-numbered stream:

  ```
  talker_2.stream = listener.stream + 1
  ```

  otherwise:

  ```
  talker_2.stream = listener.stream - 1
  ```

- Then connect:

  ```
  talker_2 -> listener_2
  ```

Making a quad connection

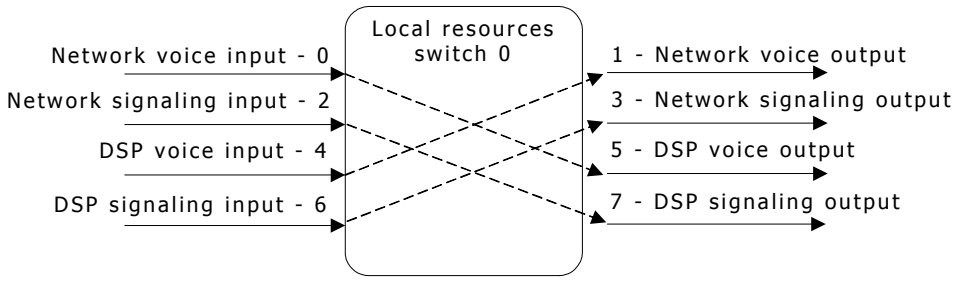
To make a quad connection, call **ppxConnect**, specify the addresses of the talker and the listener, and specify the mode PPX_QUAD. A quad connection is the same as making two duplex connections. The PPX service makes four end-to-end connections. The connections can involve the allocation of telephony bus timeslots if the talker and the listener are on different boards.

In a quad connection, the application specifies one connection, which connects a talker to a listener.

The PPX service does the following:

- Makes the duplex connection for the talker and the listener.
- Increments the streams by two, and connects a talker and listener.
- Makes the duplex connection for the new talker and listener.

The following illustration shows a quad connection between local resources on a telephony board. **ppxConnect** is called with the address of the talker (LOCAL:0:3) and the address of the listener (LOCAL:5:3). The first connection links the talker and the listener. The second connection links LOCAL:4:3 (stream 4 is the pair of stream 5) to LOCAL:1:3 (stream 1 is the pair of stream 0). The third connection links LOCAL:2:3 (stream 2 is stream 0 + 2) to LOCAL:7:3 (stream 7 is stream 5 + 2). The second connection links LOCAL:6:3 (stream 6 is the pair of stream 7) to LOCAL:3:3 (stream 3 is the pair of stream 2).



talker = 0:LOCAL:0:3
listener = 0:LOCAL:5:3

ppxConnect (***ctahd***, ***talker***, ***listener***, PPX_DUPLEX)
LOCAL:0:3 -> LOCAL:5:3
LOCAL:4:3 -> LOCAL:1:3
LOCAL:2:3 -> LOCAL:7:3
LOCAL:6:3 -> LOCAL:3:3

The PPX service makes the required connections by calling **ppxConnect** with two endpoints.

For example, connecting a network interface to the on-board DSP resources for one port of an AG 2000 consists of one PPX call. The PPX service makes four switch connections:

Application	Switch connections made by PPX service

talker.switch_number =0;
talker.bus = MVIP95_LOCAL_BUS;
talker.stream = 0;
talker.timeslot = 4;
 	
listener.switch_number = 0;
listener.bus = MVIP95_LOCAL_BUS;
listener.stream = 5;
listener.timeslot = 4;
 	
ppxConnect(ctahd, &talker, &listener, PPX_QUAD)
	talker.bus = MVIP95_LOCAL_BUS;
	talker.stream = 0;
	talker.timeslot = 4;
	
	listener.bus = MVIP95_LOCAL_BUS;
	listener.stream = 5;
	listener.timeslot = 4;
	
	swiMakeConnection(swihd, talker,listener,1)
	
	talker.bus = MVIP95_LOCAL_BUS;
	talker.stream = 4;
	talker.timeslot = 4;
	
	listener.bus = MVIP95_LOCAL_BUS;
	listener.stream = 1;
	listener.timeslot = 4;
	
	swiMakeConnection(swihd, talker,listener,1)
	
	talker.bus = MVIP95_LOCAL_BUS;
	talker.stream = 2;
	talker.timeslot = 4;
	
	listener.bus = MVIP95_LOCAL_BUS;
	listener.stream = 7;
	listener.timeslot = 4;
	
	swiMakeConnection(swihd, talker,listener,1)
	
	talker.bus = MVIP95_LOCAL_BUS;
	talker.stream = 6;
	talker.timeslot = 4;
	
	listener.bus = MVIP95_LOCAL_BUS;
	listener.stream = 1;
	listener.timeslot = 4;
	
	swiMakeConnection(swihd, talker,listener,1)

Connecting the same port on the AG 2000 to DSP resources on an AG 4000 board consists of only one PPX call, but requires eight switch connections.

Disconnecting a talker and a listener

Use **ppxDisconnect** to disconnect a previously connected simplex, duplex, or quad switch connection. Any intervening timeslots are freed. **ppxDisconnect** can breakdown only connections that are owned by the context that is associated with the connection.

Restoring initial connections

You can specify switch connections in *ppx.cfg*. These connections can be disconnected using **ppxDisconnect** if the connections were not specified with the NAILED keyword. Refer to *Specifying switch connections* on page 43 for information about connections made in *ppx.cfg*.

Use **ppxRestoreConnections** to restore the *ppx.cfg* switch connections when the client exits. If this function is not called, the switch connections are left in the state as modified by the application.

Managing groups of connections

Use the PPX service to group switch commands into a single logical transaction. This functionality creates a batch of connections that are related to each other. All operations within the transaction must be completed successfully or the transaction fails.

A group of switch operations starts with a **ppxBegin** command. Each PPX function called after **ppxBegin** returns immediately, and the switch connections are not made until **ppxSubmit** is called.

ppxSubmit is an asynchronous function, but it returns immediately. The final result of the transaction is delivered in an event. When **ppxSubmit** is called, the PPX service determines if each operation is valid and that there are valid paths for all connections.

After verifying the commands and paths, the PPX service calls the functions to perform the operations. If an error occurs when making the connections, all pending operations are aborted. The PPX service restores any of the connections made up to that point and the system is returned to its state before **ppxBegin** was called.

When all connections are made, PPXEVN_SUBMIT_COMPLETE is returned.

All PPX functions except **ppxOpenConnection** and **ppxCreateConnection** can be included in a **ppxBegin**/**ppxSubmit** block. Only one **ppxBegin**/**ppxSubmit** block can be active on a given context.

ppxBeginCancel cancels a group of commands preceded by a **ppxBegin** statement.

Terminating a PPX client

The IPC mechanism, which the PPX service uses to communicate between the PPX client and the PPX server, notifies the PPX server when a client is no longer running. When a client stops running, all connections that the client established during its lifetime are disconnected and all associated resources are freed. A PPX client does not need to perform explicit disconnections when it exits unless it is bound to a named connection.

Named connections are not automatically disconnected. If a named connection was established or opened by the exiting client, the named connection remains unchanged at the client's exit; nothing is disconnected and no resources are freed. This connection persists until explicitly destroyed at a later time by a different PPX client with **ppxDestroyNamedConnection**.

Error processing

If the PPX server encounters either PPX service errors or SWI service errors while processing a PPX function call for a client, the PPX connection database returns to its initial state at the time the function was initiated. Some PPX service functions call SWI service functions as part of normal execution.

For example, if a new talker is set for a connection, the previous talker is disconnected from the listeners on the connection, and the new talker is connected to these listeners. If the connection fails while the new talker is being connected to any one of the listening points, the previous talker is reestablished on the connection. Any new connections established up to the point of the failure are broken, and any previously existing connections to the previous talker are restored.

PPX support for NMS OAM

The PPX server can process the following NMS OAM events:

- HSWEVN_REMOVAL_REQUESTED
- OAMEVN_STOPBOARD_DONE
- CLKEVN_CONFIGURED

When the PPX server receives HSWEVN_REMOVAL_REQUESTED or OAMEVN_STOPBOARD_DONE for a board that is used in one of its initial connections (connections defined in *ppx.cfg*), it tears down the connection that is using the switch and marks the switch disabled within the fabric. Connections defined in *ppx.cfg* are the only connections that PPX acts on when it receives an NMS OAM event.

If the input on the connection (listener) is on an enabled switch, the default idle pattern is sent to the listening point. This same action is taken when a NULL talker is assigned to a connection object to which listeners have already been assigned.

When PPX receives CLKEVN_CONFIGURED, it determines whether the board's switch is being used by one of the initial connections defined in *ppx.cfg*. If it is, the connection is established. PPX also sets the state of the switch to ENABLED.

For more information on NMS OAM, refer to the *NMS OAM System User's Manual*.

This topic provides information on:

- Configuration for unpopulated CompactPCI slots
- Client connections
- Switch disabled error
- Switching service log support

Configuration for unpopulated CompactPCI slots

Boards for unpopulated CompactPCI slots are defined in *ppx.cfg*. Similarly, the following PPX configuration is needed within the switch definition of boards that are planned to be inserted into currently unpopulated CompactPCI slots:

```
SwitchState [ENABLED | DISABLED]
```

When the field is set to DISABLED, the switch is added to the switching fabric but its state is set to DISABLED. Clients receive an error if they attempt to use the switch. If this field is missing from the definition of the switch, its value is assumed ENABLED.

Client connections

Client applications are responsible for terminating connections that are effected by HSWEVN_REMOVAL_REQUESTED. PPX does not close the switch handle until all clients have disconnected from the switch. Until the switch is closed, clients can continue to make connections using the switch.

Switch disabled error

After a disabled switch is closed, users cannot make connections that reference that switch. The PPX error value PPXERR_DISABLED is returned from the following functions when a disabled switch is specified:

- **ppxAddListeners**
- **ppxConnect**
- **ppxSetTalker**

For more information, refer to *Alphabetical error summary* on page 91 and *PPX events* on page 93.

Switching service log support

If PPX's SWI command logging is configured, switch enable and disable information appear in the log file.

The following example shows the subsequent SWI log information that appears if each of the two boards (shown in the *ppx.cfg* excerpt) are removed and then reinserted.

ppx.cfg

```
[ppx]
Fabric

       # ...
       Switch 1
            SwitchType = HMIC
            DeviceName = "agsw"
            DeviceNumber = 1

       # ... etc...
       Switch 4
            SwitchType = HMIC
            DeviceName = "agsw"
            DeviceNumber = 4
       # ... etc ...

       # Initial Connections established by ppx server
       Connect
            1:local:0:0 to 1:local:17:0 QUAD
            4:local:0:2 to 4:local:17:2 QUAD
       End Connect
End Fabric
```

```
ppxswi.log
Setup of initial connections.

[Tue Jan 12 15:29:11 1999] Switch 1: Handle Opened 0x1
[Tue Jan 12 15:29:11 1999]   swiMakeConnect: hd 0x1 LOCAL:0:0 TO LOCAL:17:0
[Tue Jan 12 15:29:11 1999]   swiMakeConnect: hd 0x1 LOCAL:16:0 TO LOCAL:1:0
[Tue Jan 12 15:29:11 1999]   swiMakeConnect: hd 0x1 LOCAL:2:0 TO LOCAL:19:0
[Tue Jan 12 15:29:11 1999]   swiMakeConnect: hd 0x1 LOCAL:18:0 TO LOCAL:3:0
[Tue Jan 12 15:29:12 1999] Switch 4: Handle Opened 0x10002
[Tue Jan 12 15:29:12 1999]   swiMakeConnect: hd 0x10002 LOCAL:0:2 TO LOCAL:17:2
[Tue Jan 12 15:29:12 1999]   swiMakeConnect: hd 0x10002 LOCAL:16:2 TO LOCAL:1:2
[Tue Jan 12 15:29:12 1999]   swiMakeConnect: hd 0x10002 LOCAL:2:2 TO LOCAL:19:2
[Tue Jan 12 15:29:12 1999]   swiMakeConnect: hd 0x10002 LOCAL:18:2 TO LOCAL:3:2

Board removal

[Tue Jan 12 15:30:39 1999] Switch 4 Device 4 DISABLED
[Tue Jan 12 15:30:39 1999] Switch 4 Handle 0x10002 Closed
[Tue Jan 12 15:30:43 1999] Switch 1 Device 1 DISABLED
[Tue Jan 12 15:30:43 1999] Switch 1 Handle 0x1 Closed

Reinsert with auto reconnect

reinsert:
[Tue Jan 12 15:30:52 1999] Switch 1 Device 1 ENABLED
[Tue Jan 12 15:30:52 1999] Switch 1: Handle Opened 0x20001

reconnect:
[Tue Jan 12 15:30:52 1999]   swiMakeConnect: hd 0x20001 LOCAL:0:0 TO LOCAL:17:0
[Tue Jan 12 15:30:52 1999]   swiMakeConnect: hd 0x20001 LOCAL:16:0 TO LOCAL:1:0
[Tue Jan 12 15:30:52 1999]   swiMakeConnect: hd 0x20001 LOCAL:2:0 TO LOCAL:19:0
[Tue Jan 12 15:30:52 1999]   swiMakeConnect: hd 0x20001 LOCAL:18:0 TO LOCAL:3:0

reinsert:
[Tue Jan 12 15:30:57 1999] Switch 4 Device 4 ENABLED
[Tue Jan 12 15:30:57 1999] Switch 4: Handle Opened 0x30002

reconnect:
[Tue Jan 12 15:30:57 1999]   swiMakeConnect: hd 0x30002 LOCAL:0:2 TO LOCAL:17:2
[Tue Jan 12 15:30:57 1999]   swiMakeConnect: hd 0x30002 LOCAL:16:2 TO LOCAL:1:2
[Tue Jan 12 15:30:57 1999]   swiMakeConnect: hd 0x30002 LOCAL:2:2 TO LOCAL:19:2
[Tue Jan 12 15:30:57 1999]   swiMakeConnect: hd 0x30002 LOCAL:18:2 TO LOCAL:3:2
```

5 Specifying the switch configuration

Switch configuration overview

ppx.cfg specifies the switch configuration for the PPX service, including the following information:

- Switch fabric
- Telephony buses connected to each board
- Each board's switch definition
- Any nailed up switch connections in the system

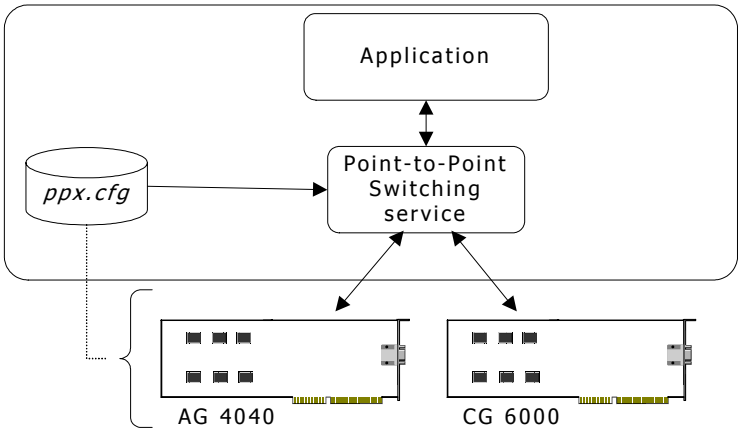
The PPX service is shipped with a template configuration file (*ppx_tmpl.cfg*) that contains examples for the various NMS board types. Copy this template to the default configuration file (*ppx.cfg*) and modify it to represent your system. You can also modify the file, give it a unique name, and specify the name when you start the PPX server using -f option.

The PPX service reads *ppx.cfg* into an internal switch connection database. When switch connections are requested, the PPX service accesses the connection database to determine how to make the desired connection.

Note: The service does not run if it cannot locate a configuration file.

The following illustration shows:

- An application requesting a connection between a DSP resource on an AG 4040 board and a line interface on an CG 6000 board.
- The PPX service accessing the connection database to determine how a connection can be made (for example, both boards are connected to the H.100 bus).
- The PPX service sending the appropriate switch commands to the boards to make the connections.



PPX configuration file syntax

The active region of the PPX configuration file begins with the string [PPX] on a line by itself and ends at the end of the file or at another word or phrase in square brackets on a line by itself. Any text outside that region is ignored by the PPX service.

White space, such as indentation and space around an equals sign, is ignored but is useful for clarity. Case is ignored. The number sign (#) and semicolon (;) are comment delimiters. The PPX service ignores any text that follows a comment delimiter character on the same line.

In the PPX configuration file, the switch fabric defines the topology of boards that are connected by telephony buses. Each computer chassis contains one switch fabric.

Defining the switch fabric

A switch fabric definition begins with a Fabric statement and ends with an End Fabric statement. All statements appearing between these two keywords define the switch fabric for the system.

A switch fabric definition has four basic components:

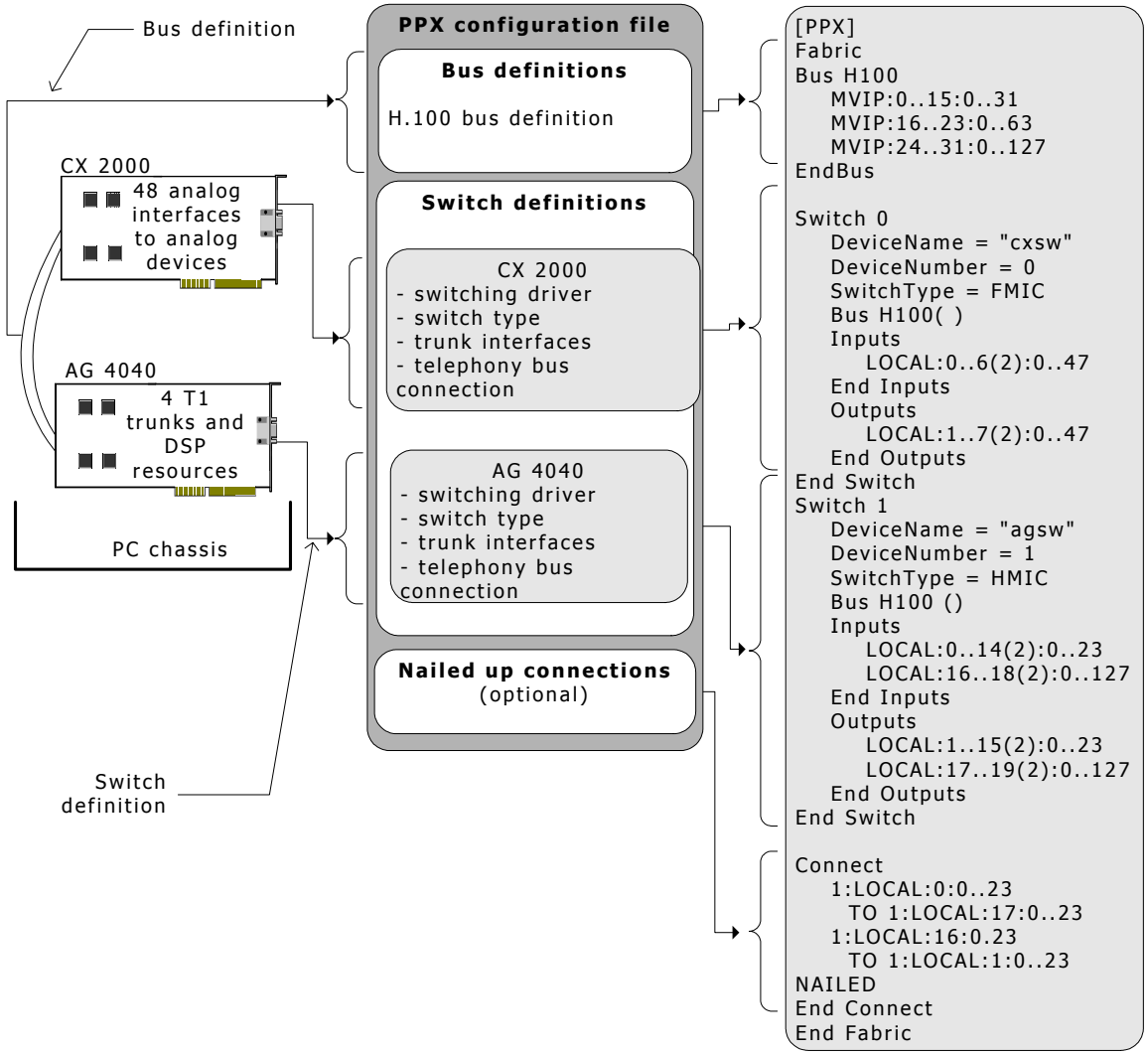
Component	Description
General attribute section	General default attributes for the switching fabric.
Bus definition section	Telephony buses in the system.
Switch configuration	Configuration of each board in the system.
Switch connections	Connections in the PPX configuration file made during PPX initialization.

The format of the PPX configuration file is:

```
[PPX]
Fabric
    # fabric attribute(s)
    # bus definition(s)
    # switch definitions
    # switch connections
End Fabric
```

All configuration information for the system must exist inside the Fabric statement.

The following illustration provides an overview of the PPX configuration file and its structure. The example system contains two boards. The PPX configuration file has a bus definition section describing the bus connecting the boards. Each board also has a switch configuration section.



Switch fabric attributes

Specify the default silence pattern used by the PPX server with the switch fabric attribute:

```
IdleCode = [MU_LAW | A_LAW]
```

This attribute must appear immediately following the configuration file keyword Fabric, but before any of the other configuration data, as the following example shows:

```
[PPX]
   Fabric
          IdleCode = MU_LAW
          # bus definition(s)
          # switch definitions
          # switch connections
End Fabric
```

If this attribute is not present within the file, the default value used is MU_LAW (0x7F).

The value of the IdleCode attribute is used as the default idle pattern for connections created with a NULL passed as the parameter pointer.

Defining the telephony bus

In the PPX configuration file, each telephony bus (H.110 or H.100) in the switch fabric must be defined and assigned a name.

Note: CT Bus is an industry standard telephony bus, defined by the ECTF H.100 and H.110 specifications. CT Bus is an interoperable superset of MVIP-90, H-MVIP, and SCbus. Therefore, references to CT Bus are inclusive of H.110, H.100, MVIP-90, and H-MVIP.

A bus is defined with either a bus specification or a bus declaration statement:

```
Bus name
    bus_specification | bus_declaration
End Bus
```

A ***bus_specification*** has the following syntax:

```
bus : streams [ (increment) ] :timeslots [ (increment)]
```

The following table describes how each element is used:

Element	Description
bus	CTBUS, for H.100 or H.110 bus.
streams	Stream assignment. Can be one stream (for example, CTBUS:0:0), multiple streams (for example, CTBUS:0,2,4:0), or a range of streams (for example, CTBUS:0..7:0).
timeslots	Timeslot assignment. Can be one timeslot (for example, CTBUS:0:0), multiple timeslots (for example, CTBUS:0:0,2,4), or a range of timeslots (for example, CTBUS:0:0..7).
increment	Specifies a value to increment (for example, CTBUS:0..4(2):0 specifies streams 0, 2, 4).

For example, a bus specification statement for the H.100 bus (8 streams at 2 Mbps; 24 streams at 8 Mbps) is:

```
Bus my_H100_bus
    CTBUS:0..7:0..31
    CTBUS:8..31:0..127
End Bus
```

A ***bus_declaration*** defines a bus as it relates to a previously defined bus and has the following syntax:

```
Bus new_bus_name
    existing_bus_name ([bus_specification])
End Bus
```

The brackets following the bus name can include a range of timeslots, indicating that the bus is a subset of another bus. For example:

```
Bus my_2MBPS_bus
    Bus my_H100_bus (CTBUS:0..7:0..31)
End Bus
```

There can be only one top-level bus definition. Specify a top-level bus with the following syntax:

```
CTBUS:streams:timeslots
```

A bus can be defined as a sub-bus of the top-level bus to indicate how buses connect. In the following example, the bus streams clocked at 2 Mbps are defined as a subset of the H.100 bus:

```
[PPX]
Fabric
    Bus my_H100_bus             # H.100 bus definition
        CTBUS:0..15:0..31       # streams clocked at 2 MHz, 32 timeslots
        CTBUS:16..23:0..63      # streams clocked at 4 MHz, 64 timeslots
        CTBUS:24..31:0..127     # streams clocked at 8 MHz, 128 timeslots
    End Bus
    Bus my_2MBPS_bus
        Bus my_H100_bus (CTBUS:0..15:0..31)
    End Bus
End Fabric
```

Reserving bus streams and timeslots

If there are specific streams and timeslots on the telephony bus that are used by another application, the streams and timeslots can be reserved by specifying a subset of timeslots in the bus definition. The reserved streams and timeslots are not available to the PPX service.

For example, to reserve the first four streams on the CT Bus, define a bus as:

```
Bus my_CT_bus
    CTBUS:0..15:0..31
End Bus
Bus PPX_AVAILABLE
    Bus my_CT_bus (CTBUS:4..15:0..31)
EndBus
```

Reference the bus, PPX_AVAILABLE, when defining the boards connected to the bus.

Defining a board's switch

Each board in the system that connects to a telephony bus must have a switch definition in the switch fabric. A section defining a particular board starts with a Switch ***n*** statement and ends with an End Switch statement.

```
Fabric
    # bus definitions for the switch fabric
    Switch 0
        # switch 0 definition
    End Switch
    Switch 1
        # switch 1 definition
    End Switch
    # switch connections
End Fabric
```

Statements inside a switch definition apply only to that particular board. Each board in the switch fabric must have a unique switch number assigned in the PPX configuration file. These switch numbers must all be unique. The numbers are arbitrary identifiers and are unrelated to the physical board configuration.

The switch-specific section for each board includes the following statements:

- Switch attributes, that describe the switch and the switching driver:

```
switch_attribute = value
```

- Telephony bus connection, that defines the bus connected to the switch on the board:

```
bus name ([bus_specification])
```

- Inputs and outputs for the board, that defines local resources, such as line interfaces and DSP resources:

```
Inputs
    bus_specification | bus_declaration
End Inputs
```

The switch-specific region for a board must include the switch attributes, telephony bus connection, and local inputs and outputs.

This topic presents:

- Defining switch attributes
- Specifying the telephony bus
- Defining local resources

Defining switch attributes

The switch attributes are:

- Switch type
- Switch driver name
- Device number of the board
- Initial switch state

The PPX service uses the switch type to determine the type of switch present on the board:

```
SwitchType = switch
```

Valid switch types include:

Switch type	Description
FMIC	Flexible MVIP interface circuit providing a complete enhanced MVIP-90 bus.
HMIC	H.100/MVIP integrated circuit, providing a complete interface to the CT Bus. **Note:** The HMIC designation is used here in a general sense; it actually represents different chips that implement the HMIC design, for example, the Lucent T8100 chip used on the AG 4000.

The switch driver name is specified by:

```
DeviceName = switch_driver
```

The device number is specified by:

```
DeviceNumber = board_number
```

The device number specified in the PPX configuration file is the board number assigned to the board. Refer to the *NMS OAM System User's Manual* for more information on board numbers.

The initial switch state is specified by:

```
SwitchState = [disabled | enabled]
```

The presence of SwitchState is optional. If it is not present, it is set to ENABLED by default. Use of this attribute is relevant to the presence of the Hot Swap service within the system. This attribute designates switches on boards that will be inserted into currently unpopulated CompactPCI slots.

The switch attributes for an AG 4040 board are shown in the following example of a PPX configuration file:

```
[PPX]
Fabric
    Bus my_H100_bus                      # bus definition
        CTBUS:0..31:0..127
    End Bus
    Switch 0                             # assigned as switch 0
        SwitchType = HMIC                # the switch on the AG 4040 is an HMIC
        DeviceName = "agsw"              # AG board device driver - agsw
        DeviceNumber = 1                 # the logical board number assigned through OAM
        # Bus declaration
        # Board inputs
        # Board outputs
    End Switch
End Fabric
```

The valid device names (switch drivers) and switch types for NMS boards are shown in the following table:

Board	Device name	Switch type
AG and CG boards	*agsw*	HMIC
QX 2000/100-4L	*qxsw*	HMIC
QX 2000/80-4L	*qxsw*	FMIC
CX 2000/C	*cxsw*	HMIC

The device number is the logical board number established with the NMS OAM board configuration.

Specifying the telephony bus

The bus declaration statement specifies the telephony bus connection to the board:

```
Bus name ([bus_specification])
```

The bus name must have been previously defined in the bus definition section of the switch fabric.

For example, the H.100 bus is defined once and assigned the name my_H100_bus. my_H100_bus is used in the switch definitions for all boards connected to the H.100 bus.

```
[PPX]
Fabric
    Bus my_H100_bus                          # H.100 bus definition
        CTBUS:0..31:0..127
    End Bus
    Switch 0                                 # assigned as switch 0
        DeviceName = "agsw"                  # AG board switch driver - agsw
        DeviceNumber = 1                     # the logical board number assigned
through OAM
        SwitchType = HMIC                    # the switch on the AG Quad is an HMIC
        Bus my_H100_bus ( )                  # AG 4040 is connected to H.100 bus
        # Board inputs
        # Board outputs
    End Switch
End Fabric
```

The brackets following the bus name can include a range of timeslots indicating that the board supports a subset of a defined bus.

In a mixed H.100 and MVIP-90 system (for example, a system with a QX 2000/100-4L and a QX 2000/80-4L) the bus definition includes all streams on the H.100 bus:

```
Bus my_H100_in_MVIP_mode
    CTBUS:0..15:0..31                       # streams clocked at 2 MHz, 32 timeslots
    CTBUS:16..23:0..63                      # streams clocked at 4 MHz, 64 timeslots
    CTBUS:24..31:0..127                     # streams clocked at 8 MHz, 128 timeslots
End Bus
```

A QX 2000/100-4L board can connect to all streams on the H.100 bus. In the switch definition for the QX 2000/100-4L board, the bus declaration is:

```
Bus my_H100_bus ()
```

The QX 2000/80-4L is an MVIP-90 board and can access only the first 16 streams. The switch definition for the QX 2000/80-4L includes a subset of the H.100 bus:

```
Bus my_H100_bus (CTBUS:0..15:0..31)
```

Defining local resources

Resources such as network interfaces and DSP resources located on the local bus as inputs to the switch (for example, voice stream coming from the telephone network) are specified by:

```
Inputs
   LOCAL:streams[(increment)]:timeslots
End Inputs
```

Resources located on the local bus as outputs from the switch (for example, voice stream going out to the telephone network) are specified by:

```
Outputs
    LOCAL:streams[(increment)]:timeslots
End Outputs
```

Typically, local resources on a board are located on consecutive streams. By convention, inputs use even-numbered streams and outputs use odd-numbered streams.

For example, for an AG 4040 board using channel associated signaling, the local resources are:

Information	Local resources
Trunk voice	Trunk 1: Streams 0 (input) and 1 (output), timeslots 0..23 Trunk 2: Streams 4 (input) and 5 (output), timeslots 0..23 Trunk 3: Streams 8 (input) and 9 (output), timeslots 0..23 Trunk 4: Streams 12 (input) and 13 (output), timeslots 0..23
Trunk signaling	Trunk 1: Streams 2 (input) and 3 (output), timeslots 0..23 Trunk 2: Streams 6 (input) and 7 (output), timeslots 0..23 Trunk 3: Streams10 (input) and 11 (output), timeslots 0..23 Trunk 4: Streams 14 (input) and 15 (output), timeslots 0..23
DSP voice	Streams 16 (input) and 17 (output), timeslots 0..127
DSP signaling	Streams 18 (input) and 19 (output), timeslots 0..127

In the PPX configuration file for the AG 4040 board, these local resources are defined as:

```
[PPX]
Fabric
    Bus my_H100_bus                          # H.100 bus definition
       CTBUS:0..31:0..127
    End Bus
    Switch 0                                 # assigned as switch 0
        DeviceName = "agsw"                  # AG board device driver - agsw
        DeviceNumber = 1                     # the logical board number assigned
through OAM
        SwitchType = HMIC                    # the switch on the AG 4040 is an HMIC
        Bus my_H100_bus ()                   # AG 4040 is connected to H.100 bus
        Inputs
            LOCAL:0..14(2):0..23             # trunk voice and signaling
            LOCAL:16..18(2):0..127           # DSP voice and signaling
        End Inputs
        Outputs
            LOCAL:1..15(2):0..23             # trunk voice and signaling
            LOCAL:17..19(2):0..127           # DSP voice and signaling
        End Outputs
    End Switch
End Fabric
```

Specifying switch connections

Specify switch connections in the PPX configuration file beginning with a Connect statement and ending with an End Connect statement. The switch connections are of the form:

```
switch_number:bus:stream:timeslot TO switch_number:bus:stream:timeslot [NAILED] mode
```

switch_number is the number assigned to the board with the Switch ***n*** statement in the PPX configuration file. ***bus*** can be CTBUS or LOCAL. ***stream*** and ***timeslot*** specify which stream and timeslot to connect.

NAILED specifies that the connections cannot be broken with PPX commands. ***mode*** can be SIMPLEX, DUPLEX, or QUAD.

For example, to connect the AG 2000 line interfaces to the on-board DSP resources:

```
[PPX]
Fabric
    Bus_2MBPS
        CTBUS:0..15:0..31
    End Bus
    Switch 0                          # AG 2000 assigned as switch 0
        DeviceName = "agsw"           # AG board device driver - agsw
        DeviceNumber = 1              # the logical board number assigned through OAM
        SwitchType = HMIC             # the switch on the AG 2000 is an HMIC
        Bus Bus_2MBPS                 # connected to the CT bus clocked at 2 mbps
        Inputs
            LOCAL:0..2(2):0..7        # trunk voice and signaling
            LOCAL:4..6(2):0..7        # DSP voice and signaling
        End Inputs
        Outputs
            LOCAL:1..3(2):0..7        # trunk voice and signaling
            LOCAL:5..7(2):0..7        # DSP voice and signaling
        End Outputs
    End Switch
    Connect
        0:LOCAL:0:0..7 TO 0:LOCAL:5:0..7 NAILED QUAD
    End Connect
End Fabric
```

Switch connections must appear as the last statements in the switch fabric definition. Connections are made when the PPX server is started. These initial switch connections can be disconnected by any application using **ppxDisconnect** unless specified with the NAILED keyword.

Sample PPX configuration file

The following PPX configuration file contains sample switch definitions for a variety of NMS boards:

```
#                       ppx.cfg
#
#          --- Point-to-Point Switching Service ---
#          ---        Configuration             ---
#
#   This file defines to the point-to-point switching service
#   the available bus, timeslots, and switches it may use for
#   establishing connection paths.
#
#   This configuration file contains example switch definitions
#   for various NMS boards. Use this as a guide for
#   configuring the PPX service to your particular hardware
#   configuration.
#
#   Some Reminders:
#       - All switch numbers must be unique.
#       - The "DeviceNumber" is the logical board number established
#         with OAM board configuration.
#       - The PPX service speaks MVIP-95, not MVIP-90.
#
#

[PPX]
Fabric
        IdleCode = MU_LAW     # Default idle pattern [ MU_LAW | A_LAW ]

        Bus H100                          # H100 bus
            CTBUS:0..15:0..31             # 2 mbps Clocking
            CTBUS:16..23:0..63            # 4 mbps "
            CTBUS:24..31:0..127           # 8 mbps "
        End Bus

         Switch 0                         # CG 6000
             SwitchType = HMIC
             DeviceName = "agsw"
             DeviceNumber = 0
             Bus H100 ()
             Inputs
                 LOCAL:0..14(2):0..23   # Trunk (0..29 for E1)
                 LOCAL:16..18(2):0..127 # DSP
             End Inputs

             Outputs
                 LOCAL:1..15(2):0..23   # Trunk (0..29 for E1)
                 LOCAL:17..19(2):0..127 # DSP
             End Outputs
        End Switch

#        Switch 1                         # CG 6100
#            SwitchType = HMIC
#            DeviceName = "agsw"
#            DeviceNumber = 1
#            Bus H100 ()
#            Inputs
#                LOCAL:0..62(2):0..29   # Trunk (0..23 for T1)
#                LOCAL:64..66(2):0..479 # DSP
#            End Inputs
#            Outputs
#                LOCAL:1..63(2):0..29   # Trunk (0..23 for T1)
#                LOCAL:65..67(2):0..479 # DSP
#            End Outputs
#        End Switch

#        Switch 2                          # AG 4040
#            SwitchType = HMIC
```

```
#            DeviceName = "agsw"
#            DeviceNumber = 2
#            Bus H100 ()
#            Inputs
#                LOCAL:0..6(2):0..23     # Trunk
#                LOCAL:16..18(2):0..127  # DSP
#            End Inputs
#            Outputs
#                LOCAL:1..7(2):0..23     # Trunk
#                LOCAL:17..19(2):0..127  # DSP
#            End Outputs
#        End Switch

#       Switch 3                         # AG 4040C
#            SwitchType = HMIC
#            DeviceName = "agsw"
#            DeviceNumber = 3
#            Bus H100 ()
#            Inputs
#                LOCAL:0..14(2):0..23    # Trunk (E1: 0..29)
#            End Inuputs
#            Outputs
#                LOCAL:1..15(2):0..23    # Trunk (E1: 0..29)
#            End Outputs
#        End Switch

#        Switch 4                        # AG 2000
#            SwitchType = HMIC
#             DeviceName = "agsw"
#             DeviceNumber = 4
#             Bus H100 ()
#             Inputs
#                 LOCAL:0..2(2):0..7     # Trunk
#                 LOCAL:4..6(2):0..7     # DSP
#             End Inputs
#             Outputs
#                 LOCAL:1..3(2):0..7     # Trunk
#                 LOCAL:5..7(2):0..7     # DSP
#             End Outputs
#         End Switch

#        Switch 5                        # AG 2000C
#            SwitchType = HMIC
#             DeviceName = "agsw"
#             DeviceNumber = 5
#             Bus H100 ()
#             Inputs
#                 LOCAL:0..2(2):0..23    # Trunk
#                 LOCAL:4..6(2):0..23    # DSP
#             End Inputs
#             Outputs
#                 LOCAL:1..3(2):0..23    # Trunk
#                 LOCAL:5..7(2):0..23    # DSP
#             End Outputs
#         End Switch

#        Switch 6                        # QX 2000/100-4L
#            SwitchType = HMIC
#             DeviceName = "qxsw"
#             DeviceNumber = 6
#             Bus H100 ()
#             Inputs
#                 LOCAL:0..6(2):0..3     # line interfaces & DSP resources
#                 LOCAL:8..10(2):0       # local phone data and signal
#                 LOCAL:12:0             # audio
#             End Inputs
#             Outputs
#                 LOCAL:1..7(2):0..3     # line interfaces & DSP resources
#                 LOCAL:9..11(2):0       # local phone data and signal
#                 LOCAL:13:0             # audio
#             End Outputs
```

```
#          End Switch

#         Switch 7                          # CX 2000C
#             SwitchType = HMIC
#              DeviceName = "cxsw"
#              DeviceNumber = 7
#              Bus H100 ()
#              Inputs
#                  LOCAL:0..2(2):0..47    # 48 station interfaces
#                  LOCAL:4..6(2):0..47    # :0..31 for CX2000C-32
#              End Inputs                 # :0..15 for CX2000C-16
#              Outputs
#                  LOCAL:1..3(2):0..47    # 48 station interfaces
#                  LOCAL:5..7(2):0..47    # :0..31 for CX2000C-32
#              End Outputs                # :0..15 for CX2000C-16
#          End Switch

#         Connect
#             0:local:0:0 to 0:local:17:0 QUAD
#             0:local:0:2 to 0:local:17:2 QUAD NAILED
#         End Connect

End Fabric
```

6 Function summary

Connection management functions

Function	Synchronous/ Asynchronous	Description
ppxCreateConnection	Synchronous	Creates a connection.
ppxOpenConnection	Synchronous	Opens a named connection and gets a handle to the connection.
ppxCloseConnection	Synchronous	Closes a connection, disconnecting any talker and/or listeners.
ppxDestroyNamedConnection	Synchronous	Destroys resources associated with a named connection.
ppxSetTalker	Synchronous	Sets the talker to a connection.
ppxAddListeners	Synchronous	Adds listeners to a connection.
ppxRemoveListeners	Synchronous	Removes listeners from a connection.
ppxSetDefaultPattern	Synchronous	Sets a default pattern for a connection.

One-talker, one-listener functions

Function	Synchronous/ Asysnchronous	Description
ppxConnect	Synchronous	Makes a simplex, duplex, or quad connection.
ppxDisconnect	Synchronous	Disconnects a simplex, duplex, or quad connection.

Group functions

Function	Synchronous/ Aysnchronous	Description
ppxBegin	Synchronous	Starts a list of PPX switching commands.
ppxSubmit	Asynchronous	Submits the list of switching commands.
ppxBeginCancel	Synchronous	Cancels a begin or submit block.

Switch reset functions

Function	Synchronous/ Asynchronous	Description
ppxRestoreConnections	Synchronous	Restores the system to the switch connections specified in the PPX configuration file.
ppxCloseSwitch	Synchronous	Forces the PPX server to close an open switch.

PPX server's runtime database function

ppxShowDB sends the contents of the PPX runtime connection database to a file.

7 Function reference

Using the function reference

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

Prototype	The prototype is shown followed by a listing of the function's arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to the list of errors returned by the Point-to-Point Switching service functions.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values is provided in the value field of the event.
Example	Example functions that start with Demo are excerpts taken from demonstration function libraries shipped with the product. Example functions that start with my are excerpts taken from sample application programs shipped with the product. The notation /* ... */ indicates additional code, which is not shown.

ppxAddListeners

Adds listeners to a connection.

Prototype

DWORD **ppxAddListeners** (PPXHD ***ppxhd***, PPX_MVIP_ADDR ***listeners[]***, unsigned ***count***)

Argument	Description
ppxhd	Handle returned by **ppxCreateConnection** or by **ppxOpenConnection**.

listeners[]	Pointer to a PPX_MVIP_ADDR structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD switch_number;`
	`  DWORD bus;`
	`  DWORD stream;`
	`  DWORD timeslot;`
	`} PPX_MVIP_ADDR`
	Refer to the Details section for a description of these fields.

count	Number of listeners to add.

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_NOT_FOUND	One of the listeners is not defined in the Point-to-Point Switching configuration file.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_DISABLED	Switch number specified was disabled.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
PPXERR_INVALID_COUNT	Number of supplied ***listeners*** is less than ***count***.
PPXERR_INVALID_SWITCH	Switch number specified is inaccessible from the specified ***bus***:***stream***:***timeslot***.
PPXERR_NO_PATH	Could not find a path from talker to one of the listeners assigned to the connection.
PPXERR_NOT_LISTENER	One of the listeners is incapable of receiving output from the connection.
SWIERR_INVALID_STREAM	One or more of the streams specified is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the timeslots specified is invalid.

Details

ppxAddListeners adds listeners to a connection. If the connection already has a talker attached, switch connections are made to connect the listeners to the talker. If the talker and listeners are on different boards, telephony bus timeslots are used to make the connection.

If no talker is attached, the listeners have the default pattern of the connection sent to them.

The PPX_MVIP_ADDR structure contains the following fields:

Field	Description
switch_number	Switch number of the board as defined in *ppx.cfg*.
bus	Acceptable values are: MVIP95_MVIP_BUS MVIP95_LOCAL_BUS
stream	The stream number of the listener.
timeslot	The timeslot of the listener.

Refer to *Adding listeners to a connection* on page 22 for more information.

See also

ppxRemoveListeners, **ppxSetDefaultPattern**, **ppxSetTalker**

Example

```
DWORD KeepOnHold( MYCONTEXT *cx )
{
    DWORD e;
    unsigned port_id;
    CTA_EVENT event;

    port_id = cx->id;

    if( cx->conn_state != CTX_ONHOLD )
    {
        /* Need to add as a listener to the hold message
         * connection.
         */

        cx->conn_state = CTX_ONHOLD;

        listeners[port_id].switch_number = cx->swi;
        listeners[port_id].bus = MVIP95_LOCAL_BUS;
        listeners[port_id].stream = 1;
        listeners[port_id].timeslot = cx->time_slot;

        e = ppxAddListeners( ppxhd[HOLD_MSG],
                             &listeners[port_id], 1 );
        if (e != SUCCESS)
        {
            return e;
        }
    }
}
```

ppxBegin

Begins a series of switching commands.

Prototype

DWORD **ppxBegin** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_INVALID_COMMAND	Another **ppxBegin** is active on the context.

Details

ppxBegin starts a series of switching commands. When **ppxSubmit** is called, the PPX service determines if each operation is valid and if there are valid paths for all connections. If the PPX service cannot make all the requested operations, the system is returned to its state before **ppxBegin** was called.

After verifying the commands and paths, the PPX service calls the functions to perform the operations. If an error occurs when making the connections, the system is returned to its state before **ppxBegin** was called.

Only one begin or submit transaction can be active on a given context.

Refer to *Managing groups of connections* on page 30 for more information.

See also

ppxBeginCancel

ppxBeginCancel

Cancels a begin or a submit transaction.

Prototype

DWORD **ppxBeginCancel** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_INVALID_COMMAND	A **ppxBegin** command was not active.

Details

ppxBeginCancel cancels a begin or a submit transaction.

See also

ppxSubmit

ppxCloseConnection

Closes a connection.

Prototype

DWORD **ppxCloseConnection** (PPXHD ***ppxhd***)

Argument	Description
ppxhd	Connection handle returned by **ppxCreateConnection** or by **ppxOpenConnection**.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxCloseConnection closes a previously created connection. If the connection is unnamed, the resources associated with the connection are freed and all listeners on the connection are returned to their default disabled states. The talker, if any, is disconnected. All telephony bus timeslots used by the connection are also freed.

If the connection is named, **ppxCloseConnection** releases only the connection handle.

Refer to *Closing a connection* on page 25 for more information.

See also

ppxDestroyNamedConnection

ppxCloseSwitch

Forces the PPX server to close an open switch.

Prototype

DWORD **ppxCloseSwitch** (CTAHD ***ctahd,*** unsigned ***swnum***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
swnum	Unsigned switch number.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	Invalid switch number.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxCloseSwitch forces the PPX server to close a switch handle (to perform a **swiCloseSwitch**). The switch connections remain in place, and the handle to the switch closes.

ppxConnect

Makes simplex, duplex, or quad switch connections.

Prototype

DWORD **ppxConnect** (CTAHD ***ctahd***, PPX_MVIP_ADDR ***talker***, PPX_MVIP_ADDR ***listener***, PPX_CX_MODE ***mode***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

talker	Pointer to a PPX_MVIP_ADDR structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD switch_number;`
	`  DWORD bus;`
	`  DWORD stream;`
	`  DWORD timeslot;`
	`} PPX_MVIP_ADDR`
	Refer to the Details section for a description of these fields.

listener	Pointer to a PPX_MVIP_ADDR structure.
mode	Connection mode: PPX_SIMPLEX, PPX_DUPLEX, or PPX_QUAD.

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_NOT_FOUND	***talker*** or ***listener*** is not defined in the PPX configuration file.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_DISABLED	Switch number specified has been disabled.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
PPXERR_INVALID_SWITCH	Switch number specified is inaccessible from the specified ***bus***:***stream***:***timeslot***.
PPXERR_LISTENER_BUSY	Specified ***listener*** is assigned to another connection.
PPXERR_NO_PATH	Could not find a path from ***talker*** to the ***listener***.
PPXERR_NOT_LISTENER	***listener*** is incapable of receiving output from the connection.
PPXERR_NOT_TALKER	Specified ***talker*** is incapable of driving the output for the connection.
SWIERR_INVALID_STREAM	One or more of the specified streams is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the specified timeslots is invalid.

Details

ppxConnect is a convenience function that wraps the functionality of **ppxCreateConnection**, **ppxSetTalker**, and **ppxAddListeners**. **ppxConnect** makes one-talker to one-listener connections. These connections use internal connection handles that cannot be accessed by the application. Listeners cannot be added to connections made by **ppxConnect**.

The PPX_MVIP_ADDR structure contains the following fields:

Field	Description
switch_number	Switch number of the board as defined in *ppx.cfg.*
bus	Acceptable values are: MVIP95_MVIP_BUS MVIP95_LOCAL_BUS
stream	The stream number of the listener/talker.
timeslot	The timeslot of the listener/talker.

Refer to *Making one-talker, one-listener connections* on page 25 for more information.

See also

ppxDisconnect

Example

```
DWORD MyConnect( CTAHD ctahd, PPX_CX_MODE mode )
{
    PPX_MVIP_ADDR listener, talker;
    DWORD ret ;

    talker.switch_number =0;
    talker.bus = MVIP95_LOCAL_BUS;
    talker.stream = 0;
    talker.timeslot = 4;

    listener.switch_number = 1 ;
    listener.bus = MVIP95_LOCAL_BUS;
    listener.stream = 5;
    listener.timeslot = 4;

     /*
      * The passed-in "mode" has one of the following
      * values:
      *
      *    PPX_SIMPLEX, PPX_DUPLEX, or PPX_QUAD
      */

    ret = ppxConnect( ctahd, &talker, &listener, mode );

    return ( ret ) ;

}
```

ppxCreateConnection

Creates a connection.

Prototype

DWORD **ppxCreateConnection** (CTAHD ***ctahd***, char ****name***, PPX_HANDLE_PARMS ****parms***, PPXHD ****ppxhd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
name	Pointer to a connection name, or NULL value.

parms	Pointer to a PPX_HANDLE_PARMS structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  BYTE default_pattern;`
	`  DWORD *reservation_key;`
	`} PPX_HANDLE_PARMS`
	Set reservation_key field to NULL. This field is not used.

ppxhd	Pointer to connection handle.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_CONNECTION_ALREADY_EXISTS	A connection already exists in the PPX connection database with the specified ***name***.

Details

ppxCreateConnection creates a connection and returns a handle to it. Initially, the connection has no talkers or listeners attached to it.

A default pattern is specified when the connection is created. All listeners added to the connection receive the default pattern if there is no talker attached. If a default pattern is not specified in the call to **ppxCreateConnection**, the default pattern is set to 0x7f (mu-law), unless otherwise specified by the IdleCode switching fabric attribute within the PPX configuration file.

A name can also be specified when creating the connection. A name is required to access the connection across applications using **ppxOpenConnection**. Named connections persist when the client exits.

Refer to *Creating a connection* on page 21 for more information.

See also

ppxCloseConnection

Example

```
void WhileTheyWait( MYCONTEXT *cx )
{
    DWORD e;
    char  ppxname [20];

    DemoStartProtocol( cx->hd, cx->protocol, NULL, NULL );

    /* Allocate a connection */
    sprintf( ppxname, "ppxdemo%01d", cx->id );
    e = ppxCreateConnection( cx->hd, ppxname, &ppx_parms,
                                  &(ppxhd[cx->id]) );
    if (e != SUCCESS)
    {
        printf("Unable to create a connection ; returns: %d\n",e);
        exit(-1);
    }
}
```

ppxDestroyNamedConnection

Frees all resources and disables outputs to all points for a named connection.

Prototype

DWORD **ppxDestroyNamedConnection** (PPXHD ***ppxhd***)

Argument	Description
ppxhd	PPX connection handle.

Return values

Return value	Description
SUCCESS	
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_NOT_FOUND	Name of the connection (referenced through ***ppxhd***) cannot be found in the PPX connection database.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxDestroyNamedConnection frees all resources and disables outputs to all points on a named connection. Once destroyed, all open handles to the named connection become invalid in all applications. Refer to *Destroying a named connection* on page 25 and *Terminating a PPX client* on page 30 for more information.

See also

ppxCreateConnection, **ppxOpenConnection**

ppxDisconnect

Disconnects an existing simplex, duplex, or quad connection.

Prototype

DWORD **ppxDisconnect** (CTAHD ***ctahd***, PPX_MVIP_ADDR ***talker***, PPX_MVIP_ADDR ***listener***, PPX_CX_MODE ***mode***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

talker	Pointer to a PPX_MVIP_ADDR structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD switch_number;`
	`  DWORD bus;`
	`  DWORD stream;`
	`  DWORD timeslot;`
	`} PPX_MVIP_ADDR`
	Refer to the Details section for a description of these fields.

listener	Pointer to a PPX_MVIP_ADDR structure.
mode	Connection mode: PPX_SIMPLEX, PPX_DUPLEX, or PPX_QUAD.

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_INVALID_HANDLE	***ctahd*** is not a valid context handle.
CTAERR_NOT_FOUND	***talker*** or ***listener*** is not defined in the PPX configuration file.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_CONN_OWNERSHIP	***listener*** is in a connection not owned by the calling connection or the initial connection was nailed up.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
PPXERR_INVALID_SWITCH	Switch number specified is inaccessible from the specified ***bus***:***stream***:***timeslot***.
PPXERR_NOT_CONNECTED	***listener*** is not receiving output from the ***talker***.
PPXERR_NOT_LISTENER	***listener*** is incapable of receiving output from the connection.
SWIERR_INVALID_STREAM	One or more of the specified streams is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the specified timeslots is invalid.

Details

ppxDisconnect disconnects a talker from a listener that was previously connected through **ppxConnect**. All intervening timeslots are freed.

The context must own the connection in order to disconnect.

If ***talker*** = NULL, the listener is disconnected (output disabled).

ppxDisconnect can also be used to disconnect any nailed up connections specified in the PPX configuration file as long as the connections were not specified with the NAILED keyword. Verify that no application is using the connection before disconnecting.

The PPX_MVIP_ADDR structure contains the following fields:

Field	Description
switch_number	Switch number of the board as defined in *ppx.cfg.*
bus	Acceptable values are: MVIP95_MVIP_BUS MVIP95_LOCAL_BUS
stream	Stream number of the listener/talker.
timeslot	Timeslot of the listener/talker.

Refer to *Making one-talker, one-listener connections* on page 25 for more information.

See also

ppxRestoreConnections

Example

```
DWORD MyConnectDisconnect( CTAHD ctahd, PPX_CX_MODE mode, BOOL MakeConn )
{
    PPX_MVIP_ADDR listener, talker;
    DWORD ret ;

    talkers.switch_number = 0;
    talker.bus = MVIP95_LOCAL_BUS;
    talker.stream = 0;
    talker.timeslot = 4;

    listener.switch_number = 1 ;
    listener.bus = MVIP95_LOCAL_BUS;
    listener.stream = 5;
    listener.timeslot = 4;

     /*
      * The passed-in "mode" has one of the following
      * values:
      *    PPX_SIMPLEX, PPX_DUPLEX, PPX_QUAD
      */
    if ( MakeConn == TRUE )

       ret = ppxConnect( ctahd, &talker, &listener, mode );

    else

       ret = ppxDisconnect( ctahd, &listener, &talker, mode );

   return ( ret ) ;
}
```

ppxOpenConnection

Opens an existing named connection.

Prototype

DWORD **ppxOpenConnection** (CTAHD ***ctahd***, char ****name***, PPX_HANDLE_PARMS ****parms***, PPXHD ****ppxhd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.
name	Pointer to the name of the connection.

parms	Pointer to a PPX_HANDLE_PARMS structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD size;`
	`  BYTE default_pattern;`
	`  DWORD *reservation_key;`
	`} PPX_HANDLE_PARMS`
	Set reservation_key field to NULL. This field is not used.

ppxhd	Pointer to connection handle.

Return values

Return value	Description
SUCCESS	
CTAERR_BAD_ARGUMENT	***name*** is either NULL or is a pointer to a string of zero length.
CTAERR_NOT_FOUND	***name*** connection can not be found in the PPX connection database.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxOpenConnection opens a named connection and returns a handle to the connection. The name of the connection was specified when creating the connection with **ppxCreateConnection**.

Any application can open a handle to a named connection and set a talker or add or remove listeners. Any action taken in one application will be apparent in every application accessing that connection. Destroy a named connection by calling **ppxDestroyNamedConnection**.

Refer to *Opening a connection* on page 24 for more information.

See also

ppxCloseConnection

ppxRemoveListeners

Removes listeners from a connection.

Prototype

DWORD **ppxRemoveListeners** (PPXHD ***ppxhd***, PPX_MVIP_ADDR ***listeners[]***, unsigned ***count***)

Argument	Description
ppxhd	Handle returned by **ppxCreateConnection** or by **ppxOpenConnection**.

listeners[]	Pointer to a PPX_MVIP_ADDR structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD switch_number;`
	`  DWORD bus;`
	`  DWORD stream;`
	`  DWORD timeslot;`
	`} PPX_MVIP_ADDR`
	Refer to the Details section for a description of these fields.

count	Number of listeners to remove.

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_NOT_FOUND	One of the ***listeners*** is not defined in the PPX configuration file.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
PPXERR_INVALID_COUNT	Number of supplied listeners is less than ***count***.
PPXERR_INVALID_SWITCH	Switch number specified is inaccessible from the specified ***bus***:***stream***:***timeslot***.
PPXERR_NOT_CONNECTED	One of the ***listeners*** is not receiving output on the connection.
PPXERR_NOT_LISTENER	One of the ***listeners*** is incapable of receiving output from the connection.
SWIERR_INVALID_STREAM	One or more of the specified streams is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the specified timeslots is invalid.

Details

ppxRemoveListeners removes listeners from a connection. All unused intervening timeslots are freed, but the talker remains associated with the connection.

The PPX_MVIP_ADDR structure contains the following fields:

Field	Description
switch_number	Switch number of the board as defined in *ppx.cfg.*
bus	Acceptable values are: MVIP95_MVIP_BUS MVIP95_LOCAL_BUS
stream	The stream number of the listener.
timeslot	The timeslot of the listener.

Refer to *Removing listeners from a connection* on page 23 for more information.

See also

ppxAddListeners

Example

```
DWORD VoiceConx( CTAHD ctahd, MYCONTEXT *cust, BOOL make_conn )
{
    DWORD         ret;

    listeners[cust->id].stream = 1;
    listeners[cust->id].timeslot = cust->time_slot;
    listeners[cust->id].switch_number = cust->swi ;
    listeners[cust->id].bus = MVIP95_LOCAL_BUS;

    if ( cust->conn_state == CTX_ONHOLD )
    {
       /*
        * If the caller had been put on hold, need to take them
        * off of hold before making the voice connection to the
        * Operator.
        */

       ppxRemoveListeners( ppxhd[HOLD_MSG], &listeners[cust->id], 1 );
    }
}
```

ppxRestoreConnections

Restores the connections specified in the configuration file.

Prototype

DWORD **ppxRestoreConnections** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxRestoreConnections restores the system to the connections specified in the PPX configuration file when the client exits. Refer to *Restoring initial connections* on page 30 for more information.

See also

ppxDisconnect

ppxSetDefaultPattern

Sets the default pattern for a connection.

Prototype

DWORD **ppxSetDefaultPattern** (PPXHD ***ppxhd***, BYTE ***pattern***)

Argument	Description
ppxhd	Handle returned by **ppxCreateConnection** or by **ppxOpenConnection**.
pattern	Default pattern (for example, 0x7F for mu-law or 0xD5 for A-law).

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
SWIERR_INVALID_STREAM	One or more of the specified streams is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the specified timeslots is invalid.

Details

ppxSetDefaultPattern sets the default pattern (8-bit fixed pattern) for a connection. If listeners are attached to the connection but no talkers, the default pattern is sent to the listeners. If listeners are connected to a connection and the talker is removed, the default pattern is sent to the listeners.

Refer to *Setting the silence pattern for a connection* on page 23 for more information.

See also

ppxAddListeners

ppxSetTalker

Sets a talker for a connection.

Prototype

DWORD **ppxSetTalker** (PPXHD ***ppxhd***, PPX_MVIP_ADDR ***talker***)

Argument	Description
ppxhd	Handle returned by **ppxCreateConnection** or by **ppxOpenConnection**.

talker	Pointer to a PPX_MVIP_ADDR structure. The structure is defined as follows:
	`typedef struct`
	`{`
	`  DWORD switch_number;`
	`  DWORD bus;`
	`  DWORD stream;`
	`  DWORD timeslot;`
	`} PPX_MVIP_ADDR`
	Refer to the Details section for a description of these fields.

Return values

Return value	Description
SUCCESS	
CTAERR_DRIVER_ERROR	Underlying driver retrieved an unrecognized error. Call **swiGetLastError** to retrieve the MVIP device error code.
CTAERR_INVALID_HANDLE	***ppxhd*** is not a valid connection handle.
CTAERR_NOT_FOUND	***talker*** is not defined in the PPX configuration file.
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_DISABLED	Specified switch number is disabled.
PPXERR_DRIVER_OPEN_FAILED	Driver open failed, or the driver DLL or device was not found.
PPXERR_INVALID_SWITCH	Switch number specified is inaccessible from the specified ***bus***:***stream***:***timeslot***.
PPXERR_NO_PATH	Could not find a path from ***talker*** to one of the listeners assigned to the connection.
PPXERR_NOT_TALKER	Specified ***talker*** is incapable of driving the output for the connection.
SWIERR_INVALID_STREAM	One or more of the specified streams is invalid.
SWIERR_INVALID_TIMESLOT	One or more of the specified timeslots is invalid.

Details

ppxSetTalker sets the talker for a connection. If listeners are attached to the connection, a switch connection is made to connect the listeners to the talker. If the talker and listeners are on different boards, telephony bus timeslots are allocated to make the connection.

If a talker is already attached to the connection, the talker is disconnected before the new talker is connected.

Specifying ***talker*** = NULL removes the previous talkers to the connection. The listeners receive the default pattern.

The PPX_MVIP_ADDR structure contains the following fields:

Field	Description
switch_number	Switch number of the board as defined in *ppx.cfg.*
bus	Acceptable values are: MVIP95_MVIP_BUS MVIP95_LOCAL_BUS
stream	The stream number of the talker.
timeslot	The timeslot of the talker.

Refer to *Setting the talker for a connection* on page 22 for more information.

See also

ppxAddListeners

Example

```
void WhileTheyWait( MYCONTEXT *cx )
{
    DWORD e;
    char  ppxname [20];

    DemoStartProtocol( cx->hd, cx->protocol, NULL, NULL );

    /* Allocate a connection */
    sprintf( ppxname, "ppxdemo%01d", cx->id );
    e = ppxCreateConnection( cx->hd, ppxname, &ppx_parms,
                                &(ppxhd[cx->id]) );
    if (e != SUCCESS)
    {
        printf("Unable to create a connection ; returns: %d\n",e);
        exit(-1);
    }

    /*
     * Set this port as the talker on this "Please hold...will try
     * that extension now" connection
     */
    talkers[cx->id].switch_number = cx->swi;
    talkers[cx->id].bus = MVIP95_LOCAL_BUS;
    talkers[cx->id].stream = dsp_stream;
    talkers[cx->id].timeslot = cx->time_slot;

    e = ppxSetTalker( ppxhd[cx->id], &talkers[cx->id] );
    if( e != SUCCESS )
    {
        printf("Could not set talker\n");
        exit(-1);
    }
}
```

ppxShowDB

Sends the contents of the PPX server runtime database to a file.

Prototype

DWORD **ppxShowDB** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.

Details

ppxShowDB causes the PPX server to send the contents of its runtime connection database to either *\nms\ctaccess\ppxdb.log* in Windows or */var/opt/nms/ctaccess/ppxdb.log* in UNIX.

A sample **ppxShowDB** output follows.

- The connection object address is provided, rather than the connection handle, since only a matching connection address shows a shared connection between clients.
- Switch numbers are not shown with the bus point (such as CTBUS:0:0) because bus points are not really owned by one particular switch.
- The line, 2:LOCAL:1:6 <= CTBUS:0:31, appears in the sample output. This means the talker point on the switch is connected to the intermediate bus point CTBUS:0:31 to drive the local point 1:6.
- A client ID of PPXSERV represents an internal context for those connections that are setup by the server based upon the configuration file.
- A client ID of 0xFFFFFFFF indicates that either the connection is an unnailed connection set up through the PPX configuration file, or it is a named connection that can be modified by any application.

Example

```
------------------------------------------------------------------
Fri Jan 30 10:08:04 2004      Natural Access PPX Runtime Database
------------------------------------------------------------------

Client Contexts Held by Context Manager
=======================================

 Client ID 0x5001001   Context State Information:
     Submit Pending 0 PPXErr 0 SWIErr 0 Restore 0 Shutdown 0

     Number of connections in use 0:
```

```
 Client ID 0x3001001   Context State Information:
     Submit Pending 0 PPXErr 0 SWIErr 0 Restore 0 Shutdown 0

     Number of connections in use 2:
     PPXHD 0x80001  ConnObj Address 0x69cf20
     PPXHD 0x90002  ConnObj Address 0x6a6c50

 Client ID 0x1001001   Context State Information:
     Submit Pending 0 PPXErr 0 SWIErr 0 Restore 0 Shutdown 0

     Number of connections in use 1:
     PPXHD 0x70001  ConnObj Address 0x69cf20

 Client ID PPXSRV  Context State Information:
     Submit Pending 0 PPXErr 0 SWIErr 0 Restore 0 Shutdown 0

     Number of connections in use 6:
     PPXHD 0x10001  ConnObj Address 0x69bb60
     PPXHD 0x20002  ConnObj Address 0x69ec50
     PPXHD 0x30003  ConnObj Address 0x69f380
     PPXHD 0x40004  ConnObj Address 0x69f9b0
     PPXHD 0x50005  ConnObj Address 0x69f770
     PPXHD 0x60006  ConnObj Address 0x69be40


Connection Objects Held by Connection Manager
=============================================

 ConnObj Address 0x69bb60  Client ID: PPXSRV  DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:0:2
                   Listener(s):    0:LOCAL:5:2

 ConnObj Address 0x69ec50  Client ID: PPXSRV  DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:4:2
                   Listener(s):    0:LOCAL:1:2

 ConnObj Address 0x69f380  Client ID: PPXSRV  DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:2:2
                   Listener(s):    0:LOCAL:7:2

 ConnObj Address 0x69f9b0  Client ID: PPXSRV  DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:6:2
                   Listener(s):    0:LOCAL:3:2

 ConnObj Address 0x69f770  Client ID: 0xffffffff DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:0:4
                   Listener(s):    0:LOCAL:5:4

 ConnObj Address 0x69be40  Client ID: 0xffffffff DefPattern 0x7f
     UNNAMED           Talker:    0:LOCAL:4:4
                   Listener(s):    0:LOCAL:1:4

 ConnObj Address 0x69cf20  Client ID: 0xffffffff DefPattern 0xff
     Name: OnHold Reference Count: 2
                   Talker:         0:LOCAL:0:6
                   Listener(s):    0:LOCAL:5:6
                                   1:LOCAL:1:6  <=  CTBUS:0:31
                                   1:LOCAL:1:4  <=  CTBUS:0:31

 ConnObj Address 0x6a6c50  Client ID: 0x3001001 DefPattern 0x7f
     UNNAMED           Talker:    1:LOCAL:0:0
                   Listener(s):    CTBUS:5:0

State of switches in Fabric:
----------------------------

     SWIHD 0x1      Switch #: 0  Dev#: 0 DevName: agsw   State: Enabled
     SWIHD 0x10002  Switch #: 1  Dev#: 0 DevName: atsdrv State: Enabled
```

ppxSubmit

Submits a series of connection calls.

Prototype

DWORD **ppxSubmit** (CTAHD ***ctahd***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext** or **ctaAttachContext**.

Return values

Return value	Description
SUCCESS	
CTAERR_OUT_OF_MEMORY	Unable to allocate memory.
CTAERR_SVR_COMM	Natural Access server communication error.
PPXERR_COMM_FAILURE	Problems were encountered when communicating with the PPX server.
PPXERR_INVALID_COMMAND	A **ppxBegin** command was not active.

Events

PPXEVN_SUBMIT_COMPLETE

Details

ppxSubmit submits a series of switching commands that are preceded with **ppxBegin**.

When **ppxSubmit** is called, the PPX service determines if each operation is valid and if valid paths exist for all connections. After verifying the commands and paths, the PPX service calls the functions to perform the operations. If an error occurs when making the connections, the system is returned to its state before **ppxBegin** was called.

The value field of PPXEVN_SUBMIT_COMPLETE contains either SUCCESS or the first non-SUCCESS response issued by the commands included in the **ppxBegin**/**ppxSubmit** block.

Only one begin or submit transaction can be active on a given context.

Refer to *Managing groups of connections* on page 30 for more information.

See also

ppxBeginCancel

8 Demonstration programs and utilities

Summary of the demonstration programs and utilities

The following demonstration programs and utilities are shipped with the PPX service:

Program or utility	Description
brd2brd	Demonstrates call transfer from an incoming line to an outgoing line over the MVIP bus. The PPX service is used to make connections.
callcntr	Demonstrates a call transfer from an incoming line to an outgoing line over the MVIP bus. The PPX service is used to create a connection and add talkers or listeners to the connection.
swish	Enables interactive or text-file-driven control of MVIP switches. *swish* provides a convenient way to manually test connections during development.
showcx95	Displays switch connections for all boards that have MVIP switches.
ppxservicecfg	Installs the PPX server as a Windows service or removes it from the registry.

Natural Access is shipped with source code for all demonstration programs. Each demonstration program is shipped as an executable program and includes its source files and its makefiles.

Before you start the demonstration programs, verify that:

- Natural Access is properly installed.
- The NMS boards are booted.
- MVIP switching is correctly configured.
- *nocc.tcp* protocol is available to the board for the outbound call.
- The PPX server is properly configured and installed.

Refer to the appropriate board manual for details on board installation.

Board to board: brd2brd

Demonstrates call transfer from an incoming line to an outgoing line over the MVIP bus using the PPX service.

Usage

```
brd2brd [options]
```

where ***options*** are:

Option	Specifies...
-b ***n***	NMS board number ***n*** for the primary incoming line. Default = 0
-B ***n***	NMS board number ***n*** for the outgoing line. Default = 1
-x ***n***	PPX switch number ***n*** for the primary incoming line. Default = 0
-X ***n***	PPX switch number ***n*** for the outgoing line. Default = 1
-s ***n***	Incoming call timeslot number ***n***. Default = 0
-S ***n***	Outgoing call timeslot number ***n***. Default = 0
-i ***protocol***	Protocol to run on the primary incoming line. Default = lps0
-o ***protocol***	Protocol to run on the outgoing line. Default = lps0

Functions

ppxConnect, **ppxDisconnect**

Description

brd2brd performs call transfer of an incoming call to an outgoing line using the PPX service. The connection is made over the CT Bus using the MVIP-95 model for stream numbering.

Procedure

This procedure assumes you are using a system with the following configuration:

- Two AG 2000 boards. One of the boards has a loop start hybrid on timeslot 4 and the other board has a DID hybrid on timeslot 2. Make sure one of the AG 2000 boards is configured to act as the bus clock master and the other is configured to act as a bus clock slave. Refer to the *AG 2000 Installation and Developer's Manual* for more information.
- The TCP files *lps0.tcp* and *wnk0.tcp* are downloaded to each of the boards.
- A 2500-type telephone is connected to the DID hybrid port, and there is a way to place and receive calls over the loop start line (such as a central-office simulator with another telephone handset connected to it).

The PPX service also requires the switching fabric to be defined in the PPX configuration file *\nms\ctaccess\cfg\ppx.cfg*. Make sure this file exists and represents the appropriate hardware configuration. The following example represents the hardware layout described here:

```
[PPX]
Fabric
  Bus H100
        CTBUS:0..15:0..31
        CTBUS:16..23:0..63
        CTBUS:24..31:0..127
  End Bus

  Switch 0                  ## AG 2000
   SwitchType = HMIC
   DeviceName = "agsw"
    DeviceNumber = 0
    Bus H100 (CTBUS:0..15:0..31)
    Inputs
        LOCAL:0..6(2):0..7
    End Inputs
    Outputs
        LOCAL:1..7(2):0..7
    End Outputs
  End Switch
 Switch 1                  ## AG 2000
   SwitchType = HMIC
   DeviceName = "agsw"
    DeviceNumber = 1
    Bus H100 (CTBUS:0..15:0..31)
    Inputs
        LOCAL:0..6(2):0..7
    End Inputs
    Outputs
        LOCAL:1..7(2):0..7
    End Outputs
  End Switch
End Fabric
```

Before you start up *brd2brd*, start *ctdaemon* and *ppxserv,* and verify that the boards are booted. Refer to the *NMS OAM System User's Manual* for information about *oamsys*, to the *Natural Access Developer's Reference Manual* for information about *ctdaemon*, and to PPX server for information about *ppxserv*.

Note: Be sure the PPX server is properly configured and installed before running *bdr2brd*. The PPX demonstration programs do not run if the server application is not running.

Complete the following steps to run *brd2brd*:

Step	Action
1	Start *ctdaemon* if it is not already running.
2	Run *oamsys* to boot the boards.
3	Start *ppxserv*.

Step	Action

4	Start *brd2brd* by entering:
	`brd2brd -b 0 -s 4 -B 1 -S 2 -i lps0 -o wnk0`
	This command indicates the following usage:
	• Board 0, timeslot 4 on the local streams for the incoming call.
	• Board 1, timeslot 2 on the local streams for the outgoing call.
	• Loop start is being run on the incoming call; wink start is being run on the outgoing call.
	As the program initializes, event messages indicate that services are open on each of the two ports. In addition, the *lps0* protocols have started on the incoming line.
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	`--------`
	`Waiting for incoming call...`
	brd2brd is now ready to receive a call.

5	Pick up the phone connected to the loop start line and dial a 3 digit number.
	brd2brd displays the following:
	`Event: ADIEVN_INCOMING_CALL`
	`Incoming Call...`
	`Answering call...`
	`Event: ADIEVN_ANSWERING_CALL`
	`Event: ADIEVN_CALL_CONNECTED, Answered`
	A greeting prompts you to provide an extension number.

6	Enter an extension number.
	brd2brd displays the following:
	`Collecting digits to dial...`
	`Event: ADIEVN_DIGIT_BEGIN, '3'`
	`Event: ADIEVN_DIGIT_END`
	`Event: ADIEVN_DIGIT_BEGIN, '3'`
	`Event: ADIEVN_DIGIT_END`
	`Event: ADIEVN_DIGIT_BEGIN, '1'`
	`Event: ADIEVN_COLLECTION_DONE,`
	`Finished digit string ='331'`
	`Got extension: 331`
	brd2brd plays *"I'll try that extension now."*
	brd2brd displays the following:
	`Playing file 'ctademo', msg #17...`
	`Playing 1 messages from 'ctademo'...`
	`Event: ADIEVN_DIGIT_END`
	`Event: VCEEVN_PLAY_DONE, Finished, msec=2240`
	`Placing a call to '331'...`
	`Event: ADIEVN_PLACING_CALL`

7	Take the outgoing phone handset off-hook and then quickly replace it to provide the needed wink. *brd2brd* displays the following: `Event: ADIEVN_CALL_CONNECTED, Signal Connected.` The program makes a duplex connection between the incoming and outgoing lines.
8	Tap on the mouthpiece of one handset to confirm that the two voice streams are connected. You should be able to hear the tapping on the other handset.

Step	Action

9	Hang up the outgoing line.
	The program tears down the call and returns to its initial state, waiting for a call. *brd2brd* displays the following:
	`Hanging up...`
	`Event: ADIEVN_CALL_RELEASED`
	`Call done.`
	`Hanging up...`
	`Event: ADIEVN_CALL_RELEASED`
	`Call done.`
	`Event: ADIEVN_STOPPROTOCOL_DONE, Finished`
	`Event: ADIEVN_STOPPROTOCOL_DONE, Finished`
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	`--------`
	`Waiting for incoming call...`

Call center: callcntr

Demonstrates call transfer from an incoming line to an outgoing line over the CT bus using the PPX service connection objects.

Usage

```
callcntr [options]
```

where ***options*** are:

Option	Specifies...
-b ***n***	NMS board number ***n*** for the incoming line(s). Default = 0
-B ***n***	NMS board number ***n*** for the outgoing line. Default = 1
-x ***n***	PPX switch number ***n*** for the primary incoming line. Default = 0
-X ***n***	PPX switch number ***n*** for the outgoing line. Default = 1
-s ***n[,n,...,n]***	From 1 to 8 timeslots for incoming lines. Default = 0
-S ***n***	Outgoing call timeslot number ***n***. Default = 0. **Note:** Timeslot ***n***+1 is used for playing the on-hold recording.
-i ***protocol***	Protocol to run on the incoming lines. All incoming lines are assumed to run the same protocol. Default = lps0
-o ***protocol***	Protocol to run on the outgoing line. Default = lps0

Functions

ppxAddListeners, **ppxConnect**, **ppxCreateConnection**, **ppxDisconnect**, **ppxRemoveListeners**, **ppxSetTalker**

Description

callcntr uses the PPX service to connect incoming and outgoing calls. This demonstration program presents two styles of point-to-point switch management:

- Single caller, single listener, that demonstrates the ease with which an incoming line can be transferred to an outgoing line over the MVIP bus.
- Single caller, multiple listeners, that demonstrates the use of a point-to-point connection.

The second mode of operation is modeled upon an operator line servicing calls that have been put on hold. This portion of the demonstration program shows the ease with which connection switches can be managed using the PPX service. The purpose is not to present a true call center application model.

In this second mode, *callcntr* creates a connection for the management of callers that have been put on hold, waiting for an operator to become available. As customers call in, and the demonstration program determines that the operator line is busy. These callers are added as listeners to the connection. These callers are later removed from the connection, one at a time, as the operator becomes available.

Note: The MVIP-95 switch model is used in *callcntr* for stream numbering.

Procedure

This procedure assumes you are using a system with the following configuration:

- Two AG 2000 boards. One of the boards has a loop start hybrid on timeslot 4 and the other board has two DID hybrids on timeslots 2 and 4. Make sure one of the AG 2000 boards is configured to act as the bus clock master and the other is configured to act as a bus clock slave. Refer to the board specific installation documentation for more information.
- The TCP files *lps0.tcp* and *wnk0.tcp* are downloaded to each of the boards.
- Two 2500-type telephones are connected to each DID hybrid port and there is a way to place and receive calls over the loop start line (such as a central office simulator with another telephone handset connected to it).

The PPX service also requires that the switching fabric be defined in the PPX configuration file *\nms\ctaccess\cfg\ppx.cfg*. Make sure this file exists and represents the appropriate hardware configuration. The following example represents the hardware layout described here:

```
[PPX]
Fabric
  Bus H100
        CTBUS:0..15:0..31
        CTBUS:16..23:0..63
        CTBUS:24..31:0..127
  End Bus

  Switch 0                  ## AG 2000
   SwitchType = HMIC
   DeviceName = "agsw"
    DeviceNumber = 0
    Bus H100 (CTBUS:0..15:0..31)
    Inputs
        LOCAL:0..6(2):0..7
    End Inputs
    Outputs
        LOCAL:1..7(2):0..7
    End Outputs
  End Switch
  Switch 1                  ## AG 2000
   SwitchType = HMIC
   DeviceName = "agsw"
    DeviceNumber = 1
    Bus H100 (CTBUS:0..15:0..31)
    Inputs
        LOCAL:0..6(2):0..7
    End Inputs
    Outputs
        LOCAL:1..7(2):0..7
    End Outputs
  End Switch
End Fabric
```

Before you start up *callcntr*, start *ctdaemon* and *ppxserv,* and verify that the boards are booted. Refer to the *NMS OAM System User's Manual* for information about OAM, to the *Natural Access Developer's Reference Manual* for information about *ctdaemon*, and to PPX server of this manual for information about *ppxserv*.

Note: Be sure the PPX server is properly configured and installed before running *callcntr*. The PPX demos fail to execute if the server application is not running.

Complete the following steps to run *callcntr*:

Step	Action
1	Start *ctdaemon* (if it is not already running).
2	Run *oamsys* to boot the boards.
3	Start *ppxserv.*

4	Start *callcntr* by entering the following command:
	`callcntr -b 1 -s 4,2 -B 0 -S 4 -i wnk0 -o lps0`
	This command indicates the following usage:
	• Board 1, timeslots 2 and 4 on the local streams for the incoming (customer) lines.
	• Board 0, timeslot 4 on the local streams for the outgoing (operator) line.
	• Board 0, timeslot 5 on the local streams for the DSP voice resource used when managing the callers on hold for an operator.
	• Wink start is being run on the incoming customer calls; loop start protocol is being run on the outgoing operator line.
	As the program initializes, event messages indicate that services are open on each of the four ports: the operator port, two customer ports, and the on hold message port. In addition, protocols are started on the two customer ports and on the hold message port.
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: CTAEVN_OPEN_SERVICES_DONE, Finished`
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	`--------`
	`Waiting for incoming call...`
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	`--------`
	`Waiting for incoming call...`
	Two messages indicate that the program is waiting for an incoming call. Each of these messages represents a thread in the demonstration program that is waiting for calls to be received on the customer lines, and indicates that a message is continually being played on the on hold port:
	`Playing file 'ctademo', msg #17...`
	`Playing 1 messages from 'ctademo'...`
	`Event: VCEEVN_PLAY_DONE, Finished ,`
	`msec=2240`
	callcntr is ready to receive an incoming call.

5	Take the handset of the telephone connected to board 1, local stream 0:4 off-hook. You hear a relay click.

6	Dial a 3-digit telephone number (such as 331) for the loop start handset.
	callcntr displays the following:
	`Event: ADIEVN_INCOMING_CALL`
	`Incoming Call...Answering call...`
	`Event: ADIEVN_ANSWERING_CALL`
	`Event: ADIEVN_CALL_CONNECTED, Answered`
	And then:
	`--------`
	`Placing a call to '331'...`
	`Event: ADIEVN_PLACING_CALL`

Step	Action
7	Answer the call on the operator phone that is connected to the loop start line when it rings. *callcntr* displays the following: `Event: ADIEVN_CALL_CONNECTED, Voice Begin Connected.` *callcntr* makes a duplex connection between the customer line and the operator line.
8	Tap on the mouthpiece of one handset to confirm that the two voice streams are connected. You hear the tapping on the other handset.
9	Take the receiver of the second telephone connected to board 1, local stream 0:2 off-hook. You hear a relay click.

10	Dial the 3-digit telephone number for the loop start handset.
	callcntr displays the following:
	`Event: ADIEVN_INCOMING_CALL`
	`Incoming Call...`
	`Answering call...`
	`Event: ADIEVN_ANSWERING_CALL`
	`Event: ADIEVN_CALL_CONNECTED, Answered`
	`Call connected.`
	Because there is a previously established connection to the operator, you continually hear a message saying *"I'll try that extension now ... I'll try that extension now ... I'll try..."*

11	Hang up both the phone originally taken off hook and the operator phone.
	callcntr displays the following message twice:
	`Hanging up...`
	`Event: ADIEVN_CALL_RELEASED`
	`Call done.`
	`Event: ADIEVN_STOPPROTOCOL_DONE, Finished`
	callcntr then displays the following message, which indicates that the customer protocol is being reestablished so that it can wait for the next call:
	`Event: ADIEVN_STARTPROTOCOL_DONE, Finished`
	Almost immediately following this message, the following message displays, indicating that a call is being placed to the operator:
	`Placing a call to '331'...`
	`Event: ADIEVN_PLACING_CALL`
	The operator phone rings.

12	Answer the call. *callcntr* displays the following: `Event: ADIEVN_CALL_CONNECTED, Voice Connected.`
13	Confirm that you no longer hear the message *"I'll try that extension now...",* but instead have a voice connection with the operator phone (See step 8).
14	Do not hang up the phones. Repeat step 10 using the phone on timeslot 4 of board 1. You hear the message as described.

Switching shell: swish

Provides interactive or text-file-driven control of MVIP and H.100 switches through the SWI service, the PPX service, or directly using the device drivers.

Usage

```
swish [-i] [filename]
```

where:

-i specifies that the default initialization file, *swish.ini*, is ignored, and ***filename*** specifies an ASCII file that contains *swish* commands to be executed before any interactive commands are executed.

Description

swish is a tool for interactive or text-file-driven control of switches. *swish* provides a convenient way to manually test connections during development to verify the commands that will be given to switches from within Natural Access applications that use the SWI service.

Procedure

Use *swish* in interactive mode, text-file-driven mode, or in a combination of the two.

In interactive mode, enter *swish* commands at the *swish* command prompt to control the switches on the underlying hardware:

```
swish
```

In interactive mode, the command prompt swish: displays. Type help to get a list of commands and arguments that *swish* supports. You can also enter the name of a command without any arguments to get more information about the command.

In text-file-driven mode, provide *swish* with the name of the text file that holds the commands to be executed. If the text file does not have the exit command at the end, *swish* goes into interactive mode after executing the commands in the text file. Use the ***filename****.swi* naming convention for the input text file.

To run *swish* using input from a text file enter the following command at the prompt:

```
swish filename
```

In text-file-driven mode, *swish* executes the commands from the specified file name. The syntax of the commands in the file is the same as the syntax of the commands in interactive mode.

swish supports four kinds of commands:

- SWI service commands
- PPX service commands
- MVIP-90 driver commands (legacy support for older systems)
- MVIP-95 driver commands.

The PPX service commands use functions from the PPX service. The names of the commands are the same as the functions provided by the PPX service, and the arguments to the commands are also similar. Refer to the function reference section for more information about the PPX service functions.

The *swish* commands use prefixes that correspond to the related functions or device drivers, as shown in the following table:

Use this prefix...	To indicate...
swi	SWI service commands.
ppx	PPX service commands.
drv	MVIP-90 driver commands (retained for legacy systems).
d95	MVIP-95 driver commands.

swish supports the following commands:

Command	Arguments	Description
ppx.CreateConnection	***ppxHd = CxName DefaultPattern***	Creates an empty connection.
ppx.OpenConnection	***ppxHd = CxName***	Opens a previously created connection.
ppx.CloseConnection	***ppxHd***	Closes a connection.
ppx.CloseX	***Switch Number***	Closes handle to opened switch.
ppx.DestroyNamedConnection	***ppxHd***	Destroys a named connection.
ppx.SetTalker	***ppxHd Talker***	Sets the talker for a connection.
ppx.AddListeners	***ppxHd ListenerList***	Adds listeners to a connection.
ppx.RemoveListeners	***ppxHd ListenerList***	Removes listeners from a connection.
ppx.SetDefaultPattern	***ppxHdpattern***	Sets the default pattern for a connection.
ppx.Connect	***TiList*** TO ***ToList*** [SIMPLEX | DUPLEX | QUAD]	Makes a connection.
ppx.Disconnect	***ToList*** FROM ***TiList*** [SIMPLEX | DUPLEX | QUAD]	Disconnects a connection.
ppx.Begin		Starts collecting commands.
ppx.BeginCancel		Cancels a begin command.
ppx.ShowDB		Dumps the PPX runtime database to a text file.
ppx.Submit		Sends commands to server.

For more information about a specific command, type the name of the command and press **Enter**.

Refer to the *Switching Service Developer's Reference Manual* for more information on the *swish* utility.

Example

The following *ppx.cfg* file is used for this example:

```
[PPX]
Fabric
  Bus H100
        CTBUS:0..15:0..31
        CTBUS:16..23:0..63
        CTBUS:24..31:0..127
  End Bus

  Switch 0                  ## AG 2000
   SwitchType = HMIC
   DeviceName = "agsw"
    DeviceNumber = 0
    Bus H100 (CTBUS:0..15:0..31)
    Inputs
        LOCAL:0..6(2):0..7
    End Inputs
    Outputs
        LOCAL:1..7(2):0..7
    End Outputs
  End Switch
End Fabric

# This SWISH script connects the Line-interfaces on an AG 2000 board as
# listeners on a connection block to a DSP resource on the AG 2000 board
# which is playing a hold message (the talker).

# Create the connection block.
ppx.CreateConnection hd0 = "" 0

# Set the talker
ppx.SetTalker hd0 0:LOCAL:4:0

# Add the listeners - add the 8 incoming lines
ppx.AddListeners hd0 0:LOCAL:0:0..7

# A new message can be played to all the listeners by simply setting a
# new talker to the connection. The previous talker is disconnected.
ppx.SetTalker hd0 0:LOCAL:4:1

# Close the connection
ppx.CloseConnection hd0
quit
```

Show switch connections: showcx95

Displays switch connections.

Usage

```
showcx95 [switch_driver]
```

Description

Displays the switch connections for all boards that have MVIP switches. If a pattern is being sent on a timeslot, the pattern value is displayed.

Procedure

To run *showcx95* for AG boards, enter the following command:

```
showcx95 agsw
```

Example

The following example makes connections between an AG 2000 board and an AG 4040 board.

The following *ppx.cfg* file is used for this example:

```
[PPX]
Fabric
    Bus H100
        CTBUS:0..15:0..31
        CTBUS:16..23:0..63
        CTBUS:24..31:0..127
    End Bus
    Switch 0                              ## AG 2000
        SwitchType = HMIC
        DeviceName = "agsw"
        DeviceNumber = 0
        Bus H100 (CTBUS:0..15:0..31)
        Inputs
            LOCAL:0..6(2):0..7
        End Inputs
        Outputs
             LOCAL:1..7(2):0..7
        End Outputs
    End Switch
    Switch 2                               ## AG 4040
        SwitchType = HMIC
        DeviceName = "agsw"
        DeviceNumber = 2
        Bus H100 ()
        Inputs
            LOCAL:0..14(2):0..23     # Trunk
            LOCAL:16..18(2):0..127   # DSP
        End Inputs
        Outputs
            LOCAL:1..15(2):0..23     # Trunk
            LOCAL:17..19(2):0..127   # DSP
        End Outputs
    End Switch
End Fabric
```

For the following *swish* commands:

```
ppx.Connect  0:local:0:0   to 2:local:17:0   SIMPLEX
ppx.Connect  0:local:0:2   to 2:local:17:2   DUPLEX
ppx.Connect  0:local:0:4   to 2:local:17:4   QUAD
```

The *showcx95* output is:

```
SHOWCX95 Version 1.1      Jan 15 2004

AGSW 0
 M-14:02                <-  L-00:00
 M-14:04                <-  L-00:02
 M-14:06                <-  L-00:04
 M-14:07                <-  L-02:04
 L-01:00                <-   0x7f
 L-01:02                <-  M-15:00
 L-01:04                <-  M-15:01
 L-01:06                <-   0x7f
 L-03:00                <-   0x00
 L-03:02                <-   0x00
 L-03:04                <-  M-15:02

AGSW 2
 M-15:00                <-  L-16:02
 M-15:01                <-  L-16:04
 M-15:02                <-  L-18:04
 L-01:00..23            <-   0x7f

[...]
 L-15:00..23            <-   0x00
 L-17:00                <-  M-14:02
 L-17:01                <-   0x7f
 L-17:02                <-  M-14:04
 L-17:03                <-   0x7f
 L-17:04                <-  M-14:06
 L-17:05..127           <-   0x7f
 L-19:00..03            <-   0x00
 L-19:04                <-  M-14:07
 L-19:05..127           <-   0x00
```

In the *showcx95* output, M indicates MVIP bus and L indicates local bus. The *showcx95* output shows three types of connections:

- Pattern 0x7F is sent to timeslot Local:01:02.

  ```
  L-01:02 <- 0x7f
  ```

- Timeslots Local:00:00..04 are writing to timeslots Local:05:00..03.

  ```
  L-05:00..03 <- L-00:00..04
  ```

- Timeslot MVIP:00:00 is writing to timeslot Local:01:00.
  Timeslot Local:00:00 is writing to timeslot MVIP:01:00 (a duplex connection).

  ```
  M{00/01}:00 <-> L{01/00}:00
  ```

Service configuration: ppxservicecfg

Installs the PPX server as a Windows service or removes it from the registry.

Usage

```
ppxservicecfg [ -install | -remove ] [options]
```

where:

-install registers the service and-remove stops the service and removes it.

and ***options*** include:

Option	Description	Default
-f ***filename***	Specifies the configuration filename.	*nms\ctaccess\cfg\ppx.cfg*
-e ***filename***	Specifies the error log file name.	*\nms\ctaccess\ppxerror.log*
-d ***filename***	Specifies the database output file name.	*\nms\ctaccess\ppxdb.log*
-l	Turns on the SWI command logging process.	OFF
-w ***number***	Specifies the line number at which the SWI log file wraps to start.	2000
-s ***filename***	Specifies the SWI command log file name.	*\nms\ctaccess\ppxswi.log*

9 Errors and events

Alphabetical error summary

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

The SWIERR_ and CTAERR_DRIVER_ERROR errors returned by some PPX functions can be returned from calls made by the PPX service to **swiSendPattern**, **swiDisableOutput**, or **swiMakeConnection**. If these errors occur, validate that the bus streams and timeslots defined within the *ppx.cfg* file are accurate in the bus definitions and within the switch definition for INPUTS and OUTPUTS.

The PPX service error and event codes are defined in the *ppxdef.h* include file. The Natural Access error codes and the SWI service error codes are defined in the *ctaerr.h and swidef.h* include files.

The following table lists the PPX service errors in alphabetical order. All errors are 32-bit.

Error name	Hexadecimal	Decimal	Description
PPXERR_CFG_FAILED	0x170004	1507334	PPX configuration file has an error.
PPXERR_COMM_FAILURE	0x17000A	1507335	Cannot communicate with PPX server.
PPXERR_CONN_OWNERSHIP	0x17000E	1507336	Attempt was made to disconnect a connection where the listener point is not owned by the calling application or the connection was nailed up.
PPXERR_CONNECTION_ALREADY_EXISTS	0x170002	1507337	Specified connection already exists.
PPXERR_DISABLED	0x17000F	1507338	Specified switch has been disabled.
PPXERR_DRIVER_OPEN_FAILED	0x17000C	1507339	Driver open failed, or the driver DLL or device was not found. Validate that the DeviceName and DeviceNumber fields are accurate in the *ppx.cfg* switch definitions.
PPXERR_INVALID_COMMAND	0x170000	1507340	Invalid command in a Begin/Submit block.
PPXERR_INVALID_COUNT	0x170009	1507341	Number of listeners in array of listeners does not match the count specified.
PPXERR_INVALID_SWITCH	0x17000B	1507342	Switch number specified is inaccessible from the specified bus, stream, and timeslot.

Error name	Hexadecimal	Decimal	Description
PPXERR_LISTENER_BUSY	0x170001	1507343	Listener is already connected to a connection block or the listener is already configured as an input.
PPXERR_NO_CFG_FILE	0x170010	1507344	Cannot locate configuration file.
PPXERR_NO_PATH	0x170005	1507345	Cannot find a connection path between the two endpoints.
PPXERR_NOT_CONNECTED	0x17000D	1507346	Listener is not receiving input from the talker.
PPXERR_NOT_LISTENER	0x170007	1507347	Listener is not capable of receiving output from the connection.
PPXERR_NOT_TALKER	0x170008	1507348	Talker is not capable of driving output to the connection.
PPXERR_UNKNOWN_SWITCHTYPE	0x170003	1507349	Switch type is invalid. Check the PPX configuration file.

Numerical error summary

The following table lists the PPX service errors in numerical order:

Hexadecimal	Decimal	Error name
0x170000	1507328	PPXERR_INVALID_COMMAND
0x170001	1507329	PPXERR_LISTENER_BUSY
0x170002	1507330	PPXERR_CONNECTION_ALREADY_EXISTS
0x170003	1507331	PPXERR_UNKNOWN_SWITCH_TYPE
0x170004	1507332	PPXERR_CFG_FAILED
0x170005	1507333	PPXERR_NO_PATH
0x170006	1507334	PPXERR_INCOMPLETE
0x170007	1507335	PPXERR_NOT_LISTENER
0x170008	1507336	PPXERR_NOT_TALKER
0x170009	1507337	PPXERR_INVALID_COUNT
0x17000A	1507338	PPXERR_COMM_FAILURE
0x17000B	1507339	PPXERR_INVALID_SWITCH
0x17000C	1507340	PPXERR_DRIVER_OPEN_FAILED
0x17000D	1507341	PPXERR_NOT_CONNECTED
0x17000E	1507342	PPXERR_CONN_OWNERSHIP
0x17000F	1507343	PPXERR_DISABLED
0x170010	1507344	PPXERR_NO_CFG_FILE

PPX events

ppxSubmit is an synchronous function that returns the event PPXEVN_SUBMIT_COMPLETE.

The value field contains either SUCCESS or the first non-SUCCESS response issued by the commands included in the **ppxBegin**/**ppxSubmit** block.

Index